Epistemic reasoning in Artificial Intelligence

Habilitation thesis

François Schwarzentruber

August 23, 2019
## II Decision problems

### 4 Model checking

4.1 Motivation ........................................... 41
4.2 A first algorithm .................................... 41
4.3 Pspace-membership ................................... 42
4.4 Other results ....................................... 46
   4.4.1 Explicit models .................................. 46
   4.4.2 Succinct models .................................. 46
4.5 Implementation ..................................... 47

### 5 Satisfiability problem

5.1 Motivation .......................................... 49
5.2 Tableau method ...................................... 49
   5.2.1 Principle ....................................... 49
   5.2.2 Tableau rules .................................... 50
5.3 The common knowledge free fragment ............... 51
5.4 Common knowledge .................................. 52
   5.4.1 Upper bound .................................... 52
   5.4.2 Reduction ....................................... 56
5.5 Arbitrary events .................................... 59
5.6 Implementation ..................................... 59

### 6 Epistemic planning

6.1 Undecidability proof ................................ 61
   6.1.1 Universal 1D cellular automata ................. 62
   6.1.2 Finite linear states ............................. 63
   6.1.3 Simulating cellular automata in DEL ............ 63
6.2 Undecidability frontier ................................ 65
6.3 Decidable restrictions ................................ 65
   6.3.1 Public actions ................................... 65
   6.3.2 Purely epistemic actions ........................ 66
6.4 Related work ....................................... 66

### 7 Propositional Epistemic planning

7.1 Introduction ........................................ 67
7.2 Preliminaries on automatic structures and decidable theories ............... 69
   7.2.1 Structures and logics ............................ 69
   7.2.2 Automatic presentations ......................... 70
7.3 Model checking against cMSO over regular automatic trees ..................... 72
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.1</td>
<td>The logic cMSO over trees</td>
<td>72</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Regular automatic trees</td>
<td>73</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Model checking against cMSO</td>
<td>75</td>
</tr>
<tr>
<td>7.4</td>
<td>Logics of knowledge and time over regular automatic trees</td>
<td>77</td>
</tr>
<tr>
<td>7.5</td>
<td>Application to epistemic planning synthesis</td>
<td>80</td>
</tr>
<tr>
<td>7.5.1</td>
<td>DEL structures</td>
<td>80</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Generalized epistemic planning and plan synthesis</td>
<td>81</td>
</tr>
<tr>
<td>7.6</td>
<td>Discussion</td>
<td>82</td>
</tr>
</tbody>
</table>

### III Applicability in AI

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Seeing and knowing</td>
<td>87</td>
</tr>
<tr>
<td>8.1</td>
<td>Inferring epistemic models</td>
<td>87</td>
</tr>
<tr>
<td>8.2</td>
<td>Agents than can move</td>
<td>88</td>
</tr>
<tr>
<td>8.3</td>
<td>Rotating cameras</td>
<td>90</td>
</tr>
<tr>
<td>8.4</td>
<td>Geometry in the syntax: poor’s man logic</td>
<td>90</td>
</tr>
<tr>
<td>8.5</td>
<td>Perspectives</td>
<td>91</td>
</tr>
<tr>
<td>9</td>
<td>Asynchronous systems</td>
<td>93</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td>93</td>
</tr>
<tr>
<td>9.2</td>
<td>Settings</td>
<td>96</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Models</td>
<td>96</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Language</td>
<td>99</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Truth conditions</td>
<td>99</td>
</tr>
<tr>
<td>9.2.4</td>
<td>Circularity</td>
<td>100</td>
</tr>
<tr>
<td>9.2.5</td>
<td>Example</td>
<td>101</td>
</tr>
<tr>
<td>9.3</td>
<td>Solving the circularity problem</td>
<td>103</td>
</tr>
<tr>
<td>9.3.1</td>
<td>When the initial model is a finite tree</td>
<td>103</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Announcing existential formulas</td>
<td>104</td>
</tr>
<tr>
<td>9.4</td>
<td>Semantic properties</td>
<td>105</td>
</tr>
<tr>
<td>9.4.1</td>
<td>Difference from PAL</td>
<td>106</td>
</tr>
<tr>
<td>9.4.2</td>
<td>Validities</td>
<td>107</td>
</tr>
<tr>
<td>9.5</td>
<td>Model checking</td>
<td>108</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Propositional announcements</td>
<td>109</td>
</tr>
<tr>
<td>9.5.2</td>
<td>Finite tree initial model</td>
<td>109</td>
</tr>
<tr>
<td>9.5.3</td>
<td>Existential announcements</td>
<td>111</td>
</tr>
<tr>
<td>9.6</td>
<td>Satisfiability for propositional announcements</td>
<td>111</td>
</tr>
<tr>
<td>9.6.1</td>
<td>Tableau method description</td>
<td>111</td>
</tr>
<tr>
<td>9.6.2</td>
<td>Tableau method soundness and completeness</td>
<td>112</td>
</tr>
<tr>
<td>9.7</td>
<td>Related work</td>
<td>116</td>
</tr>
<tr>
<td>9.7.1</td>
<td>Existing logics for asynchrony</td>
<td>117</td>
</tr>
<tr>
<td>9.7.2</td>
<td>Semi-private announcements and dynamic epistemic logic</td>
<td>117</td>
</tr>
<tr>
<td>9.7.3</td>
<td>Arbitrary public announcement logic</td>
<td>118</td>
</tr>
<tr>
<td>9.7.4</td>
<td>Distributed systems</td>
<td>118</td>
</tr>
<tr>
<td>9.8</td>
<td>Future work</td>
<td>119</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

My work is about technical results about autonomous agents (robots, humans, UAVs, etc.) that can reason about high-order knowledge properties (e.g., agent $a$ knows that agent $b$ knows...) and their dynamics. This document is based on a piece of software I developed since 2015, called Hintikka’s world and presented as a demonstration at IJCAI-ECAI 2018 [Sch18]. The tool is available at the following address:

\url{http://hintikkasworld.irisa.fr/}

Reasoning about knowledge is important for autonomous agents for taking decisions. For instance, if $a$ knows that $b$ knows whether there is fire in the building or not, and $a$ requires to know it, then $a$ may ask $b$. Morally, we consider several agents that have different views on the global state of the world and may have different goals. In other terms, our setting falls within the scope of several research subfields of computer science:

- **multi-agent systems** [Woo09] and **artificial intelligence** [MV04];
- **distributed systems** (amazingly, the number of occurrences of ‘know...’ in Michel Raynal’s book is 364 [Ray13]);
- **game theory**, in which, as Aumann pointed out [Aum99], higher-order knowledge is important.

We use the adjective **epistemic** ($\epsilonπιστηµη$ in Greek) reasoning for referring to reasoning about knowledge. In 1983, in their IJCAI tutorial [DM83], Ernest Davis and Leora Morgenstern interestingly mentioned **epistemic planning** (see Chapter 6) whereas it was not settled yet. In the rest of the introduction, we review some applications of epistemic reasoning.
1.1 Applications

1.1.1 Human-aware robots

An important challenge is to conceive robots that are able to assist humans in everyday life.

Towards socially intelligent robots

To this aim, robots need to be socially intelligent, that is to understand others and how to interact properly with others. This task is challenging and the use of robots that do not incorporate social intelligence can be quite embarrassing and awkward as reported by Barras, in *New Scientist*, vol. 204 in 2009 [Bar09]:

TUG was a hospital worker, and its colleagues expected it to have some social smarts, the absence of which led to frustration—for example, when it always spoke in the same way in both quiet and busy situations.

That is why, in order for the interaction between a robot and humans to be more natural, the robot needs to reason about higher-order knowledge, that is to have beliefs about beliefs of others ([Sca02], [BGB07], [DA16]).

**Example 1** Let us consider a situation inspired by the Sally and Anne test [WPS83]. A robot is assisting Anne and Sally at home. They are all the three in the living room. The robot sees the entire room. Anne puts the key in the drawer and goes in the kitchen for a while. Sally takes the key, uses it and put it in the box. When Anne comes back and opens the drawer, the robot tells Anne that the key is now in the box. This was possible because the robot believes that Anne believes that the key is in drawer, whereas it is in the box.

In general, robots should be able to handle false-beliefs, that is, they should understand that a human or other agents may have false information about the current world [Bo14].

Explanations

Also, we aim at developing artificial agents that can be trusted by humans. For that, these agents should be able to explain their decisions (see the notion of explainable agency, [LMSC17]). We advocate the use of formal methods and reasoning about knowledge in order to produce coherent explanations. The formal representation of knowledge is useful in several aspects:

- the robot can only explain facts it knows, and from which it can extract some reasonable proof;
- the robot should not explain a fact it believes the human already knows, as pointed out by Miller ([Mi19], p. 29).
- explanations could be given with an interaction with other agents (e.g., humans), and therefore the robot should be able to keep track of updates of knowledge of itself, knowledge of the other agents and in general higher-order epistemic properties.
1.1. APPLICATIONS

1.1.2 Distributed systems

Fagin and Halpern [HF89] advocate for the use of knowledge-based programs (e.g., ‘if a knows that b does not know x = 2 then ...’) for designing distributed systems. Also Moses claims that a necessary condition for an agent a to perform an action should be epistemic and of the form ‘a knows that...’ [Mos15].

Example 2 (computing the maximum) Let us consider some agents holding integers. The goal is that a given agent a has to print the maximum of these integers. Agent a having received the maximal value is not a condition to print it. Actually, agent a should know that he actually has the maximum value in order to print it.

The computation of knowledge could be implemented by extra local variables. For instance, a developer could proceed manually and introduce some low-level Boolean variables, e.g. $b \text{ knows } p$ for storing the Boolean value of the fact ‘b knows $p$’, $a \text{ knows } b \text{ knows } q$ for storing the Boolean value of the fact ‘a knows that b knows $q$’, etc. Nevertheless, proceeding manually would require to face some issues.

- Verification issue. How could we guarantee that the program always updates the Boolean variables ($b \text{ knows } p$, $a \text{ knows } b \text{ knows } q$, etc.) correctly?
- Design issue. What about the cost of extending the software with new features? Especially, what would be the cost of introducing new variables (e.g., a variable $\text{not } c \text{ knows } p$ for ‘c does not know $p$’ in the system)?
- Readability issue. What is the cost of understanding the program code made up of many low-level variables to capture reasoning about knowledge?

Knowledge-based programs sit at the adequate abstract level for designing distributed systems: the mechanism of acquisition of knowledge (perception, communication, etc.) is then developed independently from the decision making part. The idea is then that the real executed program is automatically generated from the knowledge-base program specification.

Concerning knowledge acquisition, gossip protocols [HHL88] are designed to spreading information among a group of agents. For instance, agents may have their time availability for a meeting and the goal is that all agents have the time availability of all agents. Another example, in a video game, is that each agent has its own information about their personal virtual character and the goal is that all agents have the information of all virtual characters.

Recently, gossip has been investigated with epistemic goals ([HM15], [CHM*16]): for instance, at the end, all agents knows that all agents have all the secrets, all agents knows that all agents knows that all agents have all the secrets, etc. A variety of epistemic gossip protocols are studied in [vDvEP*17].

1.1.3 Multi-robot systems

Multi-robot systems are in many applications more robust that a single-robot:

- monitoring buildings;
- decontaminating component parts in a nuclear installation;
- evaluating defects in different types of constructions (buildings, bridges, etc.);
- measuring pollution, noise, etc.
• automated fertilizing (especially in permaculture for which environment is complex).

In such systems, robots have imperfect information about the real state, but still need to cooperate to perform complex tasks. They need to reason about strategies in an imperfect information setting. These strategies are called uniform (see [vdHW03]). Recently, a framework with knowledge-based programs that centrally synthesized but decentralized executed has been proposed [SSZ18]. The main advantages are the readability of programs: they are more succinct than reactive policies.

1.1.4 AI and games

AI computer programs now master humans in many perfect information games, such as Chess and Go. For instance, the technique used for Go is a combination of tree search and deep neural networks [SHM+16]. Imperfect information games are even more challenging. More recently, an efficient dedicated algorithm for poker has been created [BBJT17].

More interestingly, a big challenge in AI is to design a computer program that can play arbitrary games [CLP05], without any bench of examples, but only with a description of rules (see for instance, the program Woodstock [KLP17]). The language used by the community of General Game Playing to describe game rules has recently been extended for capturing epistemic features ([Thi16], [JZPZ16]).

Higher-order knowledge plays an important role in games, especially in card games, such as Cluedo [vD02]. E.g., a player a may know that player b knows that a knows that the murderer is not Colonel Mustard and thus, play accordingly. Therefore, we claim that Dynamic Epistemic Logic can provide a suitable and rich framework to represent knowledge and actions of players.

1.1.5 Video games

Video games are typical examples of applications where artificial agents may take decisions. For scenarios to be realistic, especially in role-playing games/strategic games, it is relevant to model higher-order knowledge in that context (see [KWZ09] and [WZK08] for preliminary attempts for scripting video games with higher-order knowledge-based programs). Realistic scenarios also require to model emotions of artificial agents. Knowledge/belief is a key ingredient of some emotions, such as counterfactual emotions [LS11] (e.g., agent a regrets p if a knows she could have prevented p and a desires p to be false) and higher-order knowledge is relevant for reasoning about emotions of other agents (e.g., agent b knows that agent a regrets p).

Also, from a theoretical perspective, models that incorporate epistemic features, help to measure the intrinsic difficulty of games. For instance, Coulombe and Lynch prove some undecidability results of the existence of winning strategies in video games [CL18], meaning it is impossible to design a program that plays perfectly to some video games.

1.1.6 Psychology

Sally and Anne is a psychological test developed in [WP83] to measure the ability to imagine that others have false beliefs. The test was experimented for real [BCLF85] and autistic children seem to have difficulties, if this claim is contradicted [TF07]. To some extent, we claim that Dynamic Epistemic Logic can help to model psychological tests, implement them in tools such as Hintikka’s world, predict and explain behaviors. Interestingly, Arslan et al. ([AVTH15], [AVTH18]) develop new methods for children to improve the learning of higher-order knowledge.
1.2 Outline of the book

Part 1: Modeling

- Chapter 2 presents Dynamic epistemic logic \[\text{KvK08}\] with the software Hintikka’s world. DEL provides a logical language for expressing high-order knowledge properties (e.g. agent $a$ knows that agent $b$ knows...) and models for representing epistemic states and epistemic actions. Typically, artificial agents reason by means of an epistemic state and check properties. It corresponds to the model checking approach, promoted by Halpern and Vardi \[HV91\].

- Chapter 3 introduces succinct models for Dynamic epistemic logic. In particular, we show how to represent succinctly epistemic states and epistemic actions.

Part 2: Decision problems

- Chapter 4 presents algorithms and complexity results for the model checking problem in DEL. Model checking could be used for bounded epistemic planning when the initial state is described by means of a Kripke model. It somehow corresponds to the “situated automata” approach \[Ros85, RK86\].

- Chapter 5 focuses on the opposite approach: theorem proving. It corresponds to the standard approach in AI to knowledge representation \[McC59\]: an agent possesses a set of formulas, called a knowledge base, and she knows $\varphi$ if $\varphi$ is provable from the knowledge base. Rosenschein \[Ros85\] called this approach the “interpreted symbolic structures”. The main advantage is that a formula can embrace a set of epistemic states and not only one as in the model checking approach. Theorem proving could be used for bounded epistemic planning from any initial state satisfying a given formula.

- Chapter 6 presents an overview of (unbounded) epistemic planning. Planning is the following reachability problem: can we reach an epistemic state that satisfies a given epistemic goal? We show that it is undecidable in general and we exhibit some decidable fragments.

- Chapter 7 focuses on a large decidable fragment of epistemic planning. In this case, more involved properties that just reachability are decidable.

Part 3: Applicability in AI

The last part presents how models for epistemic states and epistemic actions may be inferred automatically in the following practical cases. In all cases, agents share common knowledge about assumptions about the system.

- Chapter 8 focuses on how epistemic states could be inferred from locations of physical agents, assuming common knowledge regarding visual abilities \[BGS13, GGS13\].

- Chapter 9 depicts how epistemic states could be inferred in an asynchronous systems, assuming common knowledge about the behavior of message passing mechanism \[KMS17\].
Part I

Modeling
Chapter 2

Dynamic epistemic logic

In this chapter, we recall basics of Dynamic epistemic logic (DEL), a logical framework for reasoning about knowledge, factual and information change. We stick to the modern presentation of DEL, that is by means of so called event models (sometimes also called action models in the literature) to model information change ([BMS98b], [vvK08], [vBvEK06]).

We will use the muddy children puzzle [vDK15] as a running example. As shown in Figure 2.1, two children are looking at each others. They know that they know that... (i.e. it is common-knowledge) that each does not know whether her forehead is muddy or not and that each knows the state of the the forehead of the other.

Let $AP$ be a countably infinite set of atomic propositions, whose typical members are denoted by $p$, $q$, etc. Atomic propositions are atomic facts in worlds, for instance ‘child a’s forehead is muddy’, and let $AGT$ be a countably infinite set of agents, whose typical members are denoted by $a$, $b$, etc. In the muddy children example, there are two agents that are the two children.

2.1 Epistemic models

In order to model an epistemic situation as shown in Figure 2.1, Saul Kripke ([Kri63], [Hin86]) introduced graphs of possible worlds, called epistemic models. It is the underlying structure used by the software Hintikka’s world. Actually, by clicking on agents in the software, it shows the unraveling of an epistemic model (see Figure 2.2 that shows the example of two muddy children [McC87]) by means of thought bubbles.

Definition 1 An epistemic model $\mathcal{M} = (W, (R_a)_{a \in AGT}, V)$ is a tuple where:

- $W$ is a non-empty finite set of possible worlds,
- $R_a \subseteq W \times W$ is an accessibility relation for agent $a$,
- $V : W \rightarrow 2^{AP}$ is a valuation function.

The intended meaning of $wR_au$ is that in world $w$ agent $a$ considers that $u$ might be the actual world. The function $V$ maps each world $w$ to the subset of atomic propositions $V(w) \subseteq AP$ that hold in $w$. A pair $(\mathcal{M}, w)$ is called a pointed epistemic model, an epistemic state, or simply a state. In a state $(\mathcal{M}, w)$, $w$ is called the real world. In Hintikka’s world, the real world is the single world shown outside any thought bubbles. We also use the notation $S$ to denote a state.
CHAPTER 2. DYNAMIC EPISTEMIC LOGIC

Possible worlds for agent \(a\) in one of the possible worlds of agent \(b\) in one of the possible worlds of agent \(a\)

Possible worlds for agent \(b\) in one of the possible worlds of agent \(a\)

Possible worlds for agent \(a\)

Real world

Figure 2.1: Muddy children with two children in Hintikka’s world.

\[
\begin{align*}
\text{Real world} & \quad \text{Possible worlds for agent } a \quad \text{Possible worlds for agent } b \quad \text{Possible worlds for agent } a
\end{align*}
\]

Figure 2.2: Epistemic model for the two muddy children.

Example 3 (Two muddy children) Figure 2.2 shows the Kripke model defined by \(\mathcal{M} = (W, (R_a)_{a \in \text{AGT}}, V)\) with

- \(W = \{w_{00}, w_{01}, w_{10}, w_{11}\}\),
- \(R_a = \{(w_{00}, w_{00}), (w_{01}, w_{01}), (w_{10}, w_{11}), (w_{00}, w_{10}), (w_{10}, w_{00}), (w_{01}, w_{11}), (w_{11}, w_{01})\}\),
- \(R_b = \{(w_{00}, w_{00}), (w_{01}, w_{01}), (w_{10}, w_{10}), (w_{11}, w_{11}), (w_{00}, w_{00}), (w_{01}, w_{01}), (w_{10}, w_{11}), (w_{11}, w_{10})\}\),
- \(V(w_{00}) = \emptyset, V(w_{01}) = \{m_b\}, V(w_{10}) = \{m_a\}, V(w_{11}) = \{m_a, m_b\}\).

The atomic propositions \(m_a\) and \(m_b\) are read as ‘agent \(a\) (resp. \(b\)) is muddy’. The set \(W\) contains the four possible configurations. The relation \(R_a\) contains the pairs of worlds of the
2.1. EPISTEMIC MODELS

form \((w_1, w_i)\), that is, pairs of worlds in which the forehead state \(i\) of agent \(b\) is the same. The relation \(R_b\) is defined symmetrically. The valuation function \(V\) informs which agents are muddy or not in a given world. Figure 2.1 depicts the state \(M,w_{11}\). The real world is \(w_{11}\), the world in which both agents \(a\) and \(b\) are muddy. The possible worlds for agent \(a\) are \(w_{11}\) and \(w_{01}\).

The two muddy children puzzle generalizes to \(n\) children. In the following example, a world is represented by the subset of agents that are the muddy ones.

Example 4 (\(n\) muddy children) The Kripke model is \(M = (W, (R_a)_{a \in AGT}, V)\) with

- \(W\) is the set of subsets of \(AGT\);
- for all agents \(a\), \(R_a = \{(w, u) \in W \times W \mid w \setminus \{a\} = u \setminus \{a\}\}\);
- \(V(w) = \{p_a \mid a \in w\}\).

The relation \(R_a\) contains pairs \((w, u)\) such that in both \(w\) and \(u\), all forehead states for all agents except \(a\) are the same. The valuation \(V(w)\) paraphrases the fact that muddy children in \(w\) are exactly those who belongs to the subset \(w\).

Example 5 (Belote) In the card game Belote, the initial configuration of the game is a possible distribution of 8 cards per player, among 4 players as shown in Figure 2.3.

Let \(AGT\) be a set of 4 players. Let \(C\) be the set of 32 cards. Such an initial configuration is a partition \(\vec{H} = (H_a)_{a \in AGT}\) of \(C\) such that for all \(a \in AGT\), \(|H_a| = 8\). The initial epistemic state corresponding to the initial configuration for the game Belote is defined by \(M = (W, (R_a)_{a \in AGT}, V)\) where:

- \(W\) is the set of possible initial configurations \(\vec{H}\);
- for all \(a \in AGT\), \(R_a = \{(\vec{H}, \vec{H}') \in W \mid H_a = H'_a\}\);
- \(V(\vec{H}) = \{p_{ac} \mid c \in H_a\}\).

The relation \(R_a\) is defined so that agent \(a\) considers another world \(\vec{H}'\) in \(\vec{H}\) iff she owns the same cards in \(\vec{H}\) and \(\vec{H}'\). The valuation simply introduces propositions \(p_{ac}\) for ‘agent \(a\) owns card \(c\)’.
Example 6 (Hanabi) Contrary to classical games, such as Belote, interestingly, in the card game Hanabi, a player sees the cards of the other players but her own cards.

Figure 2.4: A possible initial configuration of the game Hanabi.

The initial configuration of the game is a possible distribution of 5 cards per players, among 4 players as shown in Figure 2.4. Let \( AGT \) a set of 4 players. Let \( C \) be the set of 32 cards. Such an initial configuration is a partition \( \vec{H} = (H_a)_{a \in AGT} \) of a subset of \( C \) such that for all \( a \in AGT, |H_a| = 5 \). The initial epistemic state corresponding to the initial configuration for the game Belote is defined by \( M = (W, (R_a)_{a \in AGT}, V) \) where:

- \( W \) is the set of possible initial configurations \( \vec{H} \);
- for all \( a \in AGT, R_a = \{ (\vec{H}, \vec{H}') \in W \mid \text{for all } a' \in AGT \setminus \{a\}, H_{a'} = H'_{a'} \} \);
- \( V(\vec{H}) = \{ p_{ac} \mid c \in H_a \} \).

For Hanabi, the relation \( R_a \) is defined so that agent \( a \) considers another world \( \vec{H}' \) in \( \vec{H} \) possible iff she sees the same cards for the other players in \( \vec{H} \) and \( \vec{H}' \).

An epistemic model is said to be \( \text{S5} \) if the accessibility relations were equivalence relations. All the previous examples depicted \( \text{S5} \) epistemic models. The \( \text{S5} \) epistemic models are intended to model knowledge in the strict sense: capturing properties \( T, 4, 5 \) of Table 2.1. In that context, accessibility relations can be seen as undistinguishability relations: \( wR_a u \) means that agent \( a \) does not distinguish worlds \( w \) and \( u \).

<table>
<thead>
<tr>
<th>( T )</th>
<th>Agent ( a ) always consider the real world as possible</th>
<th>Reflexivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D )</td>
<td>Agent ( a ) always consider a world as possible</td>
<td>Seriality</td>
</tr>
<tr>
<td>4</td>
<td>Agent ( a ) is positively introspective</td>
<td>Transitivity</td>
</tr>
<tr>
<td>5</td>
<td>Agent ( a ) is negatively introspective</td>
<td>Euclideanity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>( wR_a u ) and ( uR_a v ) implies ( wR_a v )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( wR_a u )</td>
<td>( uR_a v )</td>
</tr>
</tbody>
</table>

Table 2.1: Relevant properties of epistemic models.

Our setting also captures belief, that is epistemic models that said to be \( \text{KD45} \) and that satisfy properties \( D, 4, 5 \) of Table 2.1. Whereas possible worlds for an agent \( a \) form an equivalence class in \( \text{S5} \) models, \( \text{KD45} \) models just exclude the real world from the set of possible worlds as shown in Figure 2.5.
2.2. EPISTEMIC LOGIC

By generality, our setting also captures weak notions of belief/knowledge for which $D$, 4 or 5 may be dropped.

Example 7 Figure 2.6 shows a state made up of two worlds $w$ and $u$. World $w$ is the real world in which all propositions are false. Agent $a$ imagines a sole possible world $u$ in which proposition $p$ is true.

Figure 2.6: Epistemic model for an agent $a$ believing $p$ whereas $p$ is false.

2.2 Epistemic logic

The language of epistemic logic, $\mathcal{L}_{EL}$, is the language of propositional logic extended with one knowledge modality for each agent. Intuitively, $K_a \varphi$ reads as “agent $a$ knows/believes that $\varphi$ holds”.

2.2.1 Syntax

The syntax of $\mathcal{L}_{EL}$ is given by the following grammar:

$$\varphi ::= p \mid \neg \varphi \mid (\varphi \lor \varphi) \mid K_a \varphi$$

where $p$ ranges over $AP$ and $a$ ranges over $AGT$. We define the usual abbreviations $(\varphi_1 \land \varphi_2)$ for $\neg(\neg \varphi_1 \lor \neg \varphi_2)$ and $\hat{K}_a \varphi$ for $\neg K_a \neg \varphi$. The size of $\varphi$ is the number of operators needed to write $\varphi$. There are three possible epistemic attitudes for agent $a$ w.r.t a formula $\varphi$:

- $K_a \varphi$: agent $a$ knows $\varphi$;
- $K_a \neg \varphi$: agent $a$ knows that $\varphi$ is false;
- $\neg K_a \varphi \land \neg K_a \neg \varphi$: agent $a$ is ignorant about the truth value of $\varphi$. 

Figure 2.5: Differences between $S5$ and $KD45$ models
Example 8 Let us consider atomic propositions $m_a$ and $m_b$ that are respectively read ‘agent $a$ is muddy’ and ‘agent $b$ is muddy’. The formula $(K_a m_b \lor K_a \neg m_b)$ is in $L_{EL}$ and is read ‘either agent $a$ knows that $b$ is muddy or agent $a$ knows that $b$ is not muddy’ or better ‘agent $a$ knows whether $b$ is muddy’.

Formula $K_a K_b((K_a m_b \lor K_a \neg m_b) \land \neg K_a m_a)$ is read ‘agent $a$ knows that agent $b$ knows that agent $a$ knows whether $b$ is muddy but agent $a$ does not know whether $a$ is muddy or not’.

2.2.2 Semantics

The semantics of $L_{EL}$ is given in terms of pointed epistemic models. We define the truth condition relation $M, w \models \varphi$ holds in the pointed epistemic model $(M, w)$.

Definition 2 $M, w \models \varphi$ is defined by induction on $\varphi$:

1. $M, w \models p$ if $p \in V(w)$;
2. $M, w \models \neg \varphi$ if it is not the case that $M, w \models \varphi$;
3. $M, w \models (\varphi \lor \psi)$ if $M, w \models \varphi$ or $M, w \models \psi$;
4. $M, w \models K_a \varphi$ if for all $u$ such that $wR_a u$, $M, u \models \varphi$.

Example 9 In the model depicted in Figure 2.2, we have:

$M, w_{11} \models K_a K_b((K_a m_b \lor K_a \neg m_b) \land \neg K_a m_a \land \neg K_a \neg m_a)$.

A formula $\varphi$ is satisfiable if there exists a pointed epistemic model $M, w$ such that $M, w \models \varphi$. A formula $\varphi$ is valid if for all pointed epistemic models $M, w$, we have $M, w \models \varphi$.

2.2.3 Common knowledge

Common knowledge $[HM90a]$ is a condition for agents to act simultaneously. Common knowledge of $\varphi$ among agents in group $G \subseteq AGT$ is the fact all agents in $G$ know that agents in $G$ know that . . . agents in $G$ know that $\varphi$. We augment the language $L_{EL}$ with the common knowledge construction $C_G \varphi$ read as ‘property $\varphi$ is commonly known by the group $G$’. The obtained language is $L_{ELCK}$, and is defined by the following grammar:

$$\varphi ::= p \mid \neg \varphi \mid (\varphi \lor \varphi) \mid K_a \varphi \mid C_G \varphi$$

where $p$ ranges over $AP$, $a$ ranges over $AGT$, and $G$ ranges over $2^{AGT}$.

The semantics of $L_{ELCK}$ is extended by adding the following clause:

1. $M, w \models C_G \varphi$ if for all $u \in W, wR_G u$ implies $M, u \models \varphi$ where $R_G$ is the transitive closure of $\bigcup_{a \in G} R_a$.

Example 10 This example just shows the relevance of common knowledge in game theory, for the so-called iterated deletion of strictly dominated strategies. Let us assume that the following normal form game is common knowledge among players $a$ and $b$. The following table gives the payoffs of the agents. For instance, if $a$ plays action $A$ and $b$ plays $\beta$, then agent $a$ receives a payoff of 4 and agent $b$ receives a payoff of 1.
2.3. EVENT MODELS

<table>
<thead>
<tr>
<th>a : A</th>
<th>b : α</th>
<th>b : β</th>
<th>b : γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : 5, b : 2</td>
<td>a : 4, b : 1</td>
<td>a : 6, b : 3</td>
<td></td>
</tr>
<tr>
<td>a : 2, b : 7</td>
<td>a : 3, b : 4</td>
<td>a : 2, b : 6</td>
<td></td>
</tr>
<tr>
<td>a : 4, b : 0</td>
<td>a : 9, b : 6</td>
<td>a : 3, b : 8</td>
<td></td>
</tr>
</tbody>
</table>

We also assume that there is common knowledge that a and b are rational. In that case, player a will actually never play action B: indeed, whatever b plays, a will always get a better payoff by playing A. In the same way, player b will never play β (do you see why?). Actually, as it is common knowledge that players are rational, it is common knowledge that B and β will never be played and we are left with the following new game:

<table>
<thead>
<tr>
<th>a : A</th>
<th>b : α</th>
<th>b : γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : 5, b : 2</td>
<td>a : 6, b : 3</td>
<td></td>
</tr>
<tr>
<td>a : 4, b : 0</td>
<td>a : 3, b : 8</td>
<td></td>
</tr>
</tbody>
</table>

Now in the same way, we see that a will not play C since it is always better to play A. For b it is always better to play γ. We finally end up with the following game:

<table>
<thead>
<tr>
<th>a : A</th>
<th>b : γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : 6, b : 3</td>
<td></td>
</tr>
</tbody>
</table>

Unfortunately, Halpern and Moses [HM90b] showed that common knowledge cannot be attained when communication may fail. To see that, they considered the so-called coordinated attack problem ([MV04], p. 64). Clark explains the role of common knowledge via the consecutive number game [Cla02]. Interestingly, the concept of common knowledge has been generalized for agents to act in different points in time: timely common knowledge [GM13].

2.3 Event models

In some mathematical models in the literature, such as epistemic temporal models [vBGP07], transition systems, concurrent game structures, decPOMDP (decentralized Partially Observable Markov Decision Processes) [BGIZ02], and interpreted systems [FMHV03b], the dynamics of systems is set in stone: the executability of an action A is directly indicated in the model – let say by the presence of an outgoing edge labelled by A from state s to s’. The effect of an action is also directly indicated by the state s’.

However, in natural languages, actions are described by their very nature (public announcements, private announcements, etc.). Let us take the example of the public announcement A of a true formula $K_a p$. We would model action A in a transition system by adding an edge from s to s’ labeled by A and we would need:

1. to check that formula $K_a p$ holds in s;
2. to guarantee that the accessibility relations for the knowledge in state s’ indeed correspond to the very knowledge of agents after the announcement of $K_a p$.

Dynamic epistemic logic (DEL) takes the perspective to describe actions; the epistemic temporal structures is then generated from these descriptions. In DEL, actions are represented by event models as proposed by Baltag et al. [BMS98b]. This very description indicates in which
states the action is executable and the effects of an action. In general DEL events can bring information and modify the world, and such events are called **ontic events** \[vDK06\]. Let us now give the definition of event models.

In this sense, DEL extends the approach taken in planning, where actions are also described. Finally, DEL extends epistemic logic with modalities that represent the occurrence of actions.

**Definition 3** An event model \(E = (E, (R^E_a)_{a \in AGT}, \text{pre}, \text{post})\) is a tuple where:

- \(E\) is a non-empty finite set of possible events,
- \(R^E_a \subseteq E \times E\) is an accessibility relation on \(E\) for agent \(a\),
- \(\text{pre} : E \rightarrow \mathcal{LELC}\) is a precondition function,
- \(\text{post} : E \times AP \rightarrow \mathcal{LELC}\) is a postcondition function.

A pair \((E, e)\) is called a **pointed event model**, where \(e\) represents the actual (real) event. A pair \((E, E_0)\) with \(E_0 \subseteq E\) is called a **multi-pointed event model**, where \(E_0\) represents the set of possible actual events. Pointed event models correspond to deterministic actions and multi-pointed event models correspond non-deterministic actions. We may confuse \((E, e)\) and \((E, \{e\})\). Sometimes, especially in Chapter 6, a pointed event model is called an action\(^1\) and is denoted by \(A\).

Interestingly, we can define intuitive restrictions on the shape of event models that correspond to different types of actions.

**Definition 4** We say that \(E\) is purely epistemic, or is without postconditions, or has no postconditions if the postcondition function is trivial in the following sense: for all event \(e\), for all atomic propositions \(p\), \(\text{post}(e, p) = p\).

**Definition 5** We say that \(E, e\) is a public action if for all agents \(a\), \((e, e) \in R^E_a\) and for all events \(e'\), \((e, e') \in R^E_a\) implies \(e' = e\).

**Definition 6** We say that \(E, e\) is a public announcement of \(\varphi\) if it is a public action, \(\text{pre}(e) = \varphi\) and if for all event \(e\), for all atomic propositions \(p\), \(\text{post}(e, p) = p\). In picture:

In words, the **public announcement** of \(\varphi\) corresponds to an event where it is common knowledge that a true message \(\varphi\) is received by all agents. The public announcement of \(\varphi\) is modelled by a pointed event model made up of a single possible event, with reflexive loops for each agent, whose precondition is \(\varphi\) and whose postcondition is trivial (truth values of atomic propositions are unchanged). In picture:

\[
\begin{array}{c}
a, b \\
\downarrow \\
\text{pre} : \varphi \\
\text{post} : / 
\end{array}
\]

The logic containing only public announcements was originally studied on its own and was proposed by \[Pla07\] earlier than the development of event models.

**Example 11 (private announcement)** The private announcement of the message \(A\) to agent \(a\)

\(^1\)We use the word *action* although according to \[aud15\], an action should always have an author.
2.3. EVENT MODELS

is modelled by the following pointed event model

Example 12 (Sally and Anne) Anne (agent $a$) moves the marble from the basket into the box while Sally (agent $b$) is outside. Proposition $p$ stands for ‘the marble is in the basket’. This event is modelled by the following pointed event model:

In the real event $e$, the marble is actually removed from the basket ($p \leftarrow \bot$) whereas the marble is not moved in the event $f$ imagined by agent $b$.

Example 13 (Learn publicly the value of $p$) It is modelled by the following multi-pointed event model:

Example 14 (Learning semi-privately the value of $p$) The fact that $a$ is learning the value of $p$ and $b$ knows that is modeled by the following multi-pointed event model:

Example 15 (Received or lost (example of Thomas Bolander)) The fact that $a$ sends the message $p$ to $b$ or that the message may be lost is:

Example 16 (child getting muddy (example of Andreas Herzig)) The fact that child $a$ is getting muddy while not knowing it, but agent $b$ is noticing it is modelled by the following pointed event model
where $m_a$ is an atomic proposition read as ‘agent $a$ is muddy’.

**Example 17 (semi-private announcement)** The semi-private announcement of the message $A$ to agent $a$ where agent $b$ knows that the message is either $A$ or $B$, while agent $a$ is not aware that agent $b$ knows it

is modelled by the following pointed event model:

2.4 Update products

In this section, we describe how to compute the updated epistemic model resulting from the execution of an action in a given epistemic model. An event $e$ can occur in a world $w$ of an epistemic model $M$ if, and only if, its precondition is verified, i.e. $M, w \models \text{pre}(e)$, which leads to the following definition:

**Definition 7** Let $M = (W, (R_a)_{a \in AGT}, V)$ be a Kripke model. Let $E = (E, (R^E_a)_{a \in AGT}, \text{pre}, \text{post})$ be an event model. The product of $M$ and $E$ is $M \otimes E = (W', (R'_a), V')$ where:

1. $W' = \{(w, e) \in W \times E \mid M, w \models \text{pre}(e)\}$;
2. $(w, e)R'_a(w', e')$ iff $wR_a w'$ and $eR^E_a e'$;
3. $V'((w, e)) = \{p \in AP \mid M, w \models \text{post}(e, p)\}$.
The product of a pointed epistemic model \((\mathcal{M}, w)\) with a pointed event model \((\mathcal{E}, e)\) is defined as \((\mathcal{M}, w) \otimes (\mathcal{E}, e) := (\mathcal{M} \otimes \mathcal{E}, (w, e))\) if \(\mathcal{M}, w \models \text{pre}(e)\), otherwise it is undefined.

An event is said to be executable in a world \(w\) if its precondition \(\text{pre}(e)\) holds in \(w\).

The update product helps to define epistemic temporal structures that correspond to the application of several update products from an initial state. They are called DEL structures but are defined later (see Definition 44). Let us point out two implicit assumptions made in DEL:

- **synchronicity**: the set of possible worlds in \(\mathcal{M} \otimes \mathcal{E}\) are all pairs; it is common knowledge that exactly one event occurred when an action is executed (see item 1 of Definition 7);
- **perfect recall** (also known as no forgetting): if \(w\) and \(w'\) are distinguished by agent \(a\), \((w, e)\) and \((w', e')\) are distinguished by agent \(a\) too (see item 2 of Definition 7); it is impossible to model an action that makes an agent forget what she learned.

### 2.5 Dynamic language

The language \(\mathcal{L}_{\text{DELCK}}\) extends \(\mathcal{L}_{\text{ELCK}}\) with dynamic modalities and is defined by the following BNF:

\[
\varphi \ ::= \top \mid p \mid \neg \varphi \mid (\varphi \lor \psi) \mid K_a \varphi \mid C_G \varphi \mid \langle \mathcal{E}, \mathcal{E}_0 \rangle \varphi
\]

with \(p \in AP\), \(a \in AGT\). Formula \(\langle \mathcal{E}, \mathcal{E}_0 \rangle \varphi\) reads as “There is an executable event in \(\mathcal{E}_0\) and \(\varphi\) holds after having executed it”. We define the dual construction \(\langle \mathcal{E}, \mathcal{E}_0 \rangle \neg \varphi\) for \(\neg \langle \mathcal{E}, \mathcal{E}_0 \rangle \neg \varphi\), that is read as “For all executable events in \(\mathcal{E}_0\), \(\varphi\) holds after having executed it”.

**Definition 8** We extend the definition \(M, w \models \varphi\) to \(\mathcal{L}_{\text{DELCK}}\) with the following clause:

- \(M, w \models \langle \mathcal{E}, \mathcal{E}_0 \rangle \varphi\) if there exists \(e \in \mathcal{E}_0\) s.th. \(M, w \models \text{pre}(e)\) and \(M \otimes \mathcal{E}, (w, e) \models \varphi\).
Example 18 Figure 2.7 shows an example of an epistemic model $M$, an event model $\mathcal{E}$ and its product $M \otimes \mathcal{E}$.

Definition 9 (public announcement logic) When $\mathcal{E}, e$ is the public announcement of $\varphi$, we write $\langle \varphi! \rangle$ instead of $\langle \mathcal{E}, e \rangle$. The language with dynamic operators that are public announcements and without common knowledge by $L_{ELPA}$. The language with dynamic operators that are public announcements and with common knowledge by $L_{ELCKPA}$.

Table 2.2 sums up the different languages considered in this book.

<table>
<thead>
<tr>
<th>Knowledge operators $K_a$</th>
<th>$L_{EL}$</th>
<th>$L_{ELCK}$</th>
<th>$L_{ELPA}$</th>
<th>$L_{DEL}$</th>
<th>$L_{DELCK}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common knowledge $C_G$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Public announcement</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dynamic operators $\langle \varphi! \rangle$</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>General dynamic operators $\langle \mathcal{E}, e \rangle$</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.2: Logical languages of epistemic logic.

2.6 Bisimulation and equivalence

We define notions of equivalence for models. The main motivation for bisimulation is, given an epistemic model for a given situation, to compute an equivalent model that is smaller. Figure 2.8 shows two pointed epistemic model that are actually equivalent (as we will see) and the model on the right is smaller than the one on the left.

\[ \begin{array}{c}
\text{Figure 2.8: Two bisimilar pointed epistemic models.}
\end{array} \]

2.6.1 Bisimulations for epistemic models

The usual notion of equivalence for epistemic models is bisimulation.

Definition 10 Let $AP$ be a finite set of atomic propositions, $M = (W, (R_a)_{a \in AGT}, V)$ and $M' = (W', (R'_a)_{a \in AGT}, V')$ be two Kripke models, $w \in W$ and $w' \in W'$. A relation $B \subseteq W \times W'$ is a $AP$-bisimulation between $M$ and $M'$ iff for all $w \in W, w' \in W'$ such that $wBw'$:
2.6. BISIMULATION AND EQUIVALENCE

- Invariance: \( V(w)_{|AP} = V'(w')_{|AP} \);
- Zig: for all \( a \in AGT \), for all \( u \in W \) such that \( wR_au \) there exists \( u' \in W' \) such that \( w'R'_au' \) and \( uBu' \);
- Zag: for all \( a \in AGT \), for all \( u' \in W' \) such that \( w'R'_au' \) there exists \( u \in W \) such that \( wR_au \) and \( uBu' \).

We say that two pointed Kripke models \((M, w)\) and \((M', w')\) are AP-bisimilar if there is an AP-bisimulation \( B \) between \( M \) and \( M' \) with \( wBw' \). \((M, w)\) and \((M', w')\) are AP-bisimilar iff \( ((M, w) \models \varphi \iff (M', w') \models \varphi) \) for all formulas \( \varphi \) of \( \mathcal{L}_{\text{DEL}} \), whose propositions are in \( AP \). When \( AP \) is clear from the context, we say bisimilar instead of AP-bisimilar. Two Kripke models \( M \) and \( M' \) are bisimilar if there exist \( w \in W \) and \( w' \in W' \) such that \((M, w)\) and \((M', w')\) are bisimilar.

**Example 19** Figure 2.8 shows two bisimilar pointed Kripke models. A bisimulation \( B \) is

\[
\{(w_0, w'_0), (w, w'_0), (u, u')\}.
\]

2.6.2 Equivalence of event models

For event models, the equivalence is defined as follows.

**Definition 11** Let \((E, e)\) and \((E', e')\) be two pointed event models. They are equivalent if for all pointed Kripke models \((M, w)\), \((M \otimes E, (w, e))\) and \((M \otimes E', (w, e'))\) are bisimilar.

Equivalence of event models without postconditions is characterized by action emulations (\(\text{VERS12}\)) defined as follows.

**Definition 12** Let \( E = (E, (-\rightarrow)_a^{E})_{a \in AGT, \text{pre}} \) and \( E' = (E', (-\rightarrow)^{E'})_{a \in AGT, \text{pre'}} \) be two event models without postconditions. Let \( \Sigma \) be the set of preconditions appearing in \( E \) and \( E' \). Let \( \hat{\Sigma} \) be the set of formulas containing \( \Sigma \) and closed under sub-formulas and negation (see \(\text{VERS12}\) for more details). Let \( CS(\hat{\Sigma}) \) be the set of maximal consistent subsets of \( \hat{\Sigma} \). An action emulation \( AE \) is a set of relations \( \{AE_{\Gamma}\}_{\Gamma \in CS(\hat{\Sigma})} \subseteq E \times E' \) such that whenever \( eAE_{\Gamma} e' \):

- Invariance: \( \text{pre}(e) \in \Gamma \) and \( \text{pre}(e') \in \Gamma' \);
- Zig: For all \( f \in E \) and \( \Gamma' \in CS(\hat{\Sigma}) \), if \( e \rightarrow^{E}_a f \), \( \text{pre}(f) \in \Gamma' \) and the formula \( \left( \Lambda^\psi_{\psi \in \Gamma} \right) \left( \Lambda^\psi_{\psi \in \Gamma'} \right) \) is consistent, then there exists \( f' \in E' \) such that \( e' \rightarrow^{E'}_a f' \) and \( fAE_{\Gamma} f' \);
- Zag: symmetric of Zig for \( E' \).

Action emulation is similar to bisimulation in the sense that the types of rules are the same, except that instead of imposing equivalence for preconditions, we just ask here that they are in a same maximal consistent subset of \( \hat{\Sigma} \).

Action emulation characterizes equivalence: \((E, e)\) and \((E', e')\) are equivalent if there is an action emulation \( AE \) between \( E \) and \( E' \) such that for all and \( \Gamma \in CS(\hat{\Sigma}) \) such that \( \text{pre}(e) \in \Gamma \), we have \( eAE_{\Gamma} e' \).
2.7 Quantification over actions

In the version of dynamic epistemic logic we presented in this chapter, we can reason about specific instances of actions, for instance, the public announcement of $p$, the public announcement of $q$, etc. In contrast, we may want to reason about action types, for instance, the class of all public announcements. Several logics for reasoning about the existence of a specific action of an action type have been developed. Instead of reasoning about a particular action described by an action model, we reason about the existence of an action of a given type. Arbitrary public announcement logic \[ \mathcal{FvD}08 \] provides a construction $\Diamond \psi$ read as ‘there exists a ($\Diamond$-free) formula $\varphi$ that is true and such that $\psi$ after announcing $\varphi$’.

Group announcement logic is a variant of arbitrary public announcement logic \[ \mathcal{ABvDS}10 \], that provides constructions $\Diamond G \psi$ where $G$ is a group of agents, for which the announced formula $\varphi$ is of the form $\bigwedge_{a \in G} K_a \varphi_a$.

Refinement modal logic \[ \mathcal{BvDF}^{+14} \] corresponds to the logic for reasoning about arbitrary action models with trivial postconditions (the correspondence is treated by Prop. 14 in \[ \mathcal{BvDF}^{+14} \], and Prop 4, 5 in \[ \mathcal{vDF}08 \]).

Aucher proposed a language to describe a class of action models by a formula over a similar language than epistemic logic \[ \mathcal{Auc}11 \].

2.8 Related work

2.8.1 Variant of epistemic models

The model used in robotics by Devin and Alami \[ \mathcal{DA}16 \] corresponds to Kripke structures where epistemic relations are functional. In other terms, every agent $a$ has a single possible world that she imagines to be the real one. The same approach was used in \[ \mathcal{BGB}07 \]. Although this restriction is reasonable for Sally and Anne, it is not relevant for more complex combinatorial cases, e.g. card games.

Dynamic epistemic logic, as presented in this chapter does not capture belief revision \[ \mathcal{AGM}85 \]; an announcement of $\varphi$ while an agent believes $\neg \varphi$ will delete all possible worlds for that agent.

Baltag and Smets extends DEL to obtain models for capturing belief revision \[ \mathcal{BS}06 \]. Some authors distinguish implicit and explicit beliefs \([\mathcal{Lev}84, \mathcal{FH}87, \mathcal{Ve}14\]) . Lorini proposed models for implicit and explicit beliefs that are hybrid: they involve Kripke semantics and syntactic sets of formulas \[ \mathcal{Lo}18 \].

2.8.2 Interpreted systems

Interpreted systems \[ \mathcal{FMHV}03b \] are an alternative to model distributed systems. Possible worlds are runs of the system and two runs are indistinguishable for agent $a$ iff agent $a$ has the same local state in both runs. Contrary to DEL, the setting of interpreted systems also captures asynchronous systems. However, the semantics is grounded: epistemic relations are not given but inferred, and are equivalence relations.

2.8.3 Action language vs. transition systems

In the planning community, the description of actions is central. That is why, specification languages, as STRIPS \[ \mathcal{FN}71 \] and PDDL, enable to describe a collection of actions. Dynamic epistemic logic follows this very same tradition. However, the fact that epistemic situations and actions are both described by graphs is neat and powerful in terms of expressivity: higher-order updates are made very precise by event models. Also the description of actions is independent
from the states they are applied in. Only the semantics of an action is dependent of the current state: the behavior of the announcement of \( \varphi \) depends on the truth values of \( \varphi \) in each possible world. The language mAL proposed in \cite{BGPS13} is a dynamic extension of epistemic logic with constructions to express observability, awareness, communication, aimed at solving the so-called ramification problem. Thielscher proposed an epistemic extension of Game Description Language (\cite{Thi16}, \cite{Thi17}) by adding epistemic and perception constructions to the language.

On the contrary, in the verification community, they also tackle imperfect information, but actions themselves are not central and the system (or each part of the system) is described globally by a transition system. It is the case in concurrent game structures or in decPOMDPs (decentralized Partially Observable Markov Decision Processes) \cite{BGIZ02}. Typically, a transition is labeled by a joint action of agents and observations. The observations are atomic; at first glance higher-order knowledge seems not important. However, even for these kind of models, higher-order knowledge plays a role for conciseness, as shown in \cite{SSZ15}. It makes the framework less flexible: there is no description of actions; they are directly hard-coded in the transition system. For instance, it is unclear how to model a public announcement of a given formula \( \varphi \): it would require to compute the truth value of \( \varphi \) and to add the corresponding transitions. Interestingly, DEL and ETL (epistemic temporal logic) are compared in \cite{vBGP07}.

Importantly, we insist on the fact that having actions described in the logical languages enables to figure out syntactic fragments on the type of actions: public announcements, public actions, restrictions on the modal depth of preconditions, ontic actions, etc.

### 2.8.4 Learning models

Gierasimczuk proposes algorithms for learning event models from samples \cite{Gic09}.

### 2.8.5 Variant of event models

Similar alternatives to event models exist: arrow update \cite{KR11}, edge-conditioned event models \cite{Bol14}. Analogously to the decomposition of integers as a product of prime numbers, an event model can decomposed. First, an event model (without postconditions) is equivalent up to modal depth \( n \) to a program made up of instances of so-called test event model, non-determinism, sequence and a learning operation \cite{FHT14}. Similarly, an event model can decomposed in terms of so-called copy and remove operators \cite{AvDFS17}.

Interestingly, Aucher studies the logical properties when epistemic models are multi-pointed, in order to capture internal perspective of agents \cite{Auc12}. 
Chapter 3

Succinct representation

In classical DEL, epistemic models and event models are represented explicitly. It faces some practical limits when models become so large they cannot be stored in the memory of a computer.

Example 20 (Belote) The initial epistemic model for Belote (see Example 5) contains
\[
\binom{32}{8} \binom{24}{8} \binom{16}{8} = 9.95 \times 10^{16} \text{ worlds.}
\]

Example 21 (Hanabi) The initial epistemic model for Hanabi (see Example 6) contains
\[
\binom{50}{5} \binom{45}{5} \binom{40}{5} \binom{35}{5} = 5.5 \times 10^{23} \text{ worlds.}
\]

Also it may be exponential in the number of atomic propositions and/or the number of agents.

Example 22 (muddy children) The initial epistemic model for muddy children contains \(2^n\) worlds, where \(n\) is the number of agents.

In order to be able to automatically reason about such large models, we propose to define succinct models for DEL, i.e. succinct epistemic models and event models. Such models were originally introduced in [CS15] and [CS17], but we present the simplification of the definitions proposed in [CS18] compared to their original definitions. In particular, the presentation of the new definitions of succinct epistemic models and succinct event models are neater: their sets of atomic propositions and not mixed as in [CS17].

3.1 Accessibility programs

Instead of describing the epistemic relations \(R^e_M\) and \(R^e_D\) respectively in epistemic models and event models in extension, we describe them in intention by using accessibility programs. Technically, we use Dynamic Logic with Propositional Assignments proposed by Herzig et al. ([BHST14], [BHT13]). Use will become clear when succinct epistemic models are presented. First, let us recall the syntax and semantics of accessibility programs.

Definition 13 The syntax for accessibility programs is defined by the BNF
\[
\pi ::= p \leftarrow \beta \mid \beta ? \mid (\pi; \pi) \mid (\pi \cup \pi)
\]
where \(p \in AP\), \(\beta\) is a Boolean formula.
Program $p \leftarrow \beta$ reads as “assign atomic proposition $p$ to the truth value of $\beta$”. Program $\beta ?$ reads as “test $\beta$”. Program $\pi_1; \pi_2$ reads as “execute $\pi_1$ then $\pi_2$”. Program $(\pi_1 \cup \pi_2)$ reads as “either execute $\pi_1$ or $\pi_2$”.

**Definition 14** The semantics of $\pi$ is the binary relation over valuations defined by induction on $\pi$ as follows, where $w$ and $u$ are valuations:

- $w \xrightarrow{p \leftarrow \beta} u$ if $(u = w \setminus \{p\}$ and $w \not\models \beta$) or $(u = w \cup \{p\}$ and $w \models \beta$);
- $w \xrightarrow{\beta ?} u$ if $w = u$ and $w \models \beta$;
- $w \xrightarrow{\pi_1; \pi_2} u$ if there exists a valuation $v$ such that $w \xrightarrow{\pi_1} v$ and $v \xrightarrow{\pi_2} u$;
- $w \xrightarrow{\pi_1 \cup \pi_2} u$ if $w \xrightarrow{\pi_1} u$ or $w \xrightarrow{\pi_2} u$;

We distinguish a particular program $\text{assign}(p_1, ..., p_n)$ that non-deterministically assigns values to $p_1, ..., p_n$, namely:

$$\text{assign}(p_1, ..., p_n) = (p_1 \leftarrow \bot \cup p_1 \leftarrow \top); \dots; (p_n \leftarrow \bot \cup p_n \leftarrow \top)$$

We now give examples of programs.

**Example 23 (Programs for muddy children)** Since child $a$ sees the forehead of $b$ but not her own, the program for $a$ amounts to modifying the truth value of $p_a$. In other terms, to obtain a possible world for child $a$, from the current world, we non-deterministically change the value of proposition $p_a$. That is, the program that succinctly describe the epistemic relation for agent $a$ is $\pi_a = \text{assign}(p_a)$. Symmetrically $\pi_b = \text{assign}(p_b)$.

**Example 24 (Program for an omniscient agent)** The program $\top ?$ represents an omniscient agent since the only reachable valuation with $\top ?$ is the actual valuation, the only possible world is the current one. For instance in the muddy children, the program of the father is $\pi_f = \top ?$.

**Example 25 (Program for an agent believing $p$)** Let $AP = \{p, q\}$. The program $\pi = p \leftarrow \top; \text{assign}(q)$ represents an agent $a$ that believes $p$ but is uncertain of $q$.

The size of an accessibility program corresponds to the number of operators needed to write the program. For instance, the mental program $(p \leftarrow \top) \cup (q?; p \leftarrow \bot)$ has size 10. The models are succinctly described by means of mental programs. We are interested here in describing epistemic models, event models and the product update.

## 3.2 Succinct epistemic models

From now on, we suppose that we have a set $AP$ to define the formulas. We now give the definition of succinct epistemic models.

### 3.2.1 Definition

**Definition 15** A succinct epistemic model is a tuple $\mathfrak{M} = \langle AP_M, \beta_M, (\pi_a)_{a \in AGT}\rangle$ where $AP_M \supseteq AP$ is a finite set of atomic propositions, $\beta_M$ is a Boolean formula over $AP_M$, and $\pi_a$ is a program over $AP_M$ for each agent $a$.

The Boolean formula $\beta_M$ succinctly describes the set of epistemic states. Intuitively, each $\pi_a$ succinctly describes the accessibility relation $\rightarrow_a$ for an agent $a$. A pointed succinct epistemic model is a pair $\mathfrak{M}, w$ where $\mathfrak{M} = \langle AP_M, \beta_M, (\pi_a)_{a \in AGT}\rangle$ is a succinct epistemic model and $w$ is a valuation satisfying $\beta_M$. 
3.2. SUCCINCT EPISTEMIC MODELS

3.2.2 From succinct epistemic models to epistemic models

Definition 16 Given a succinct epistemic model $\mathcal{M} = (\mathcal{AP}_M, \beta_M, (\pi_a)_{a \in AGT})$, the epistemic model represented by $\mathcal{M}$, noted $\mathcal{M}(\mathcal{M})$, is the model $\mathcal{M} = (W, (R_a)_{a \in AGT}, V)$ where $W = \{ w \in V(\mathcal{AP}_M) | w \models \beta_M \}$; $R_a = \{ (w, u) \in W^2 | w \overset{a}{\rightarrow} u \}$; $V(w) = w$.

Example 26 In the muddy children example [McC87], each child does not know whether she is muddy or not, but knows the muddiness of the other children. If $m_a$ is a proposition for “child $a$ is muddy”, a succinct epistemic model is $\mathcal{M} = (\mathcal{AP}_M, \beta_M, (\pi_a)_{a \in AGT})$ with $\mathcal{AP}_M = \{ m_a, a \in AGT \}$, $\beta_M = \top$ and $\pi_a = \text{assign}(m_a)$. This representation is polynomial in the number of agents $|AGT|$ whereas the non-succinct epistemic model has exponential size in $|AGT|$. Figure 3.1 shows $\mathcal{M}(\mathcal{M})$ for two agents $a$ and $b$.

3.2.3 From Kripke models to succinct models

Any epistemic model can be represented as a succinct Kripke model of polynomial size in the size of the epistemic model, in the worst case. To do so, we define a succinct Kripke model $\mathcal{M}_\mathcal{M}$ representing the Kripke model $\mathcal{M}$ with respect to a set of propositions $\mathcal{AP}$.

Definition 17 Let $\mathcal{M} = (W, (R_a)_{a \in AGT}, V)$ be a Kripke model. We define the succinct Kripke model $\mathcal{M}_\mathcal{M} = (\mathcal{AP}_M, \beta_M, (\pi_a)_{a \in AGT})$ where: $\mathcal{AP}_M = \mathcal{AP} \cup \{ p_w | w \in W \}$; $\beta_M = \exists! \{ p_w | w \in W \} \land \bigwedge_{w \in W} p_w \rightarrow \text{desc}(V(w)); \pi_a = \bigcup_{w, R_a u, p_w} \text{assign}(\mathcal{AP}_M); p_u ?$.

The intended meaning of the fresh atomic propositions $p_w$ is to designate the world $w$ (as nominals in hybrid logic [Bla00]). Formula $\beta_M$ describes the set $W$ and the valuation $V$. The formula $\exists! \{ p_w | w \in W \}$ is a Boolean formula that says that there is a unique proposition $p_w$ that is true. Program $\pi_a$ performs a non-deterministic choice over edges $w \rightarrow_a u$ and then simulate the transition $w \rightarrow_a u$. The following proposition states that $\mathcal{M}_\mathcal{M}$ indeed represents $\mathcal{M}$.

Proposition 1 $\mathcal{M}(\mathcal{M}_\mathcal{M}), \{ p_w \} \cup V(w)$ and $\mathcal{M}, w$ are $\mathcal{AP}$-bisimilar.

Proof.

We note $\mathcal{M} = (W, (\rightarrow_a)_{a \in AGT}, V)$ and $\mathcal{M}(\mathcal{M}_\mathcal{M}) = (W', (\rightarrow'_a)_{a \in AGT}, V')$. We define $B := \{ (u, p_w \cup V(u)) | u \in W \}$. Let us prove that $B$ is a $\mathcal{AP}$-bisimulation. Invariance of $B$ is routine. For Zig, consider $w \in W$. For all $u \in W$, if $w \rightarrow_a u$ then we can verify that $\mathcal{M}(\mathcal{M}_\mathcal{M}), w \cup V(u) \models \beta_M$ and that $p_w \cup V(u) \overset{a}{\rightarrow} p_u \cup V(u)$ so we have $w \rightarrow'_a u$. The Zag property is symmetrical. ■
3.2.4 Succinctness of succinct Kripke models

To show the succinctness of succinct Kripke models, we describe a family of models having exponentially more compact representations with succinct Kripke models.

Example 28 We consider the extension of the classical muddy children puzzle [DKI12] where children may pay attention to the father or not (as in [BvDH16]). We introduce an atomic proposition $m_a$ meaning that $a$ is muddy and atomic proposition $h_a$ meaning that agent $a$ hears (i.e., pays attention to) the announcements of the father. We consider the model $M = (W, (\rightarrow_a)_{a \in \text{AGT}}, \nu)$ where $W = \nu((m_a, h_a | a \in \text{AGT}))$, $\nu \rightarrow_a u$ if $(\nu \models h_a \iff \nu \models h_a)$ and for all $b \neq a$ $(\nu \models m_b \iff \nu \models m_b)$, and $V(\nu) = \nu$ for all $\nu \in W$.

In $M$, an agent $a$ will not distinguish two worlds $\nu$ from $u$ as long he sees the same forehead states for the other agents and his pay attention status is the same in both worlds.

Let us consider the family of models $M_n$ given in Example 28 for all numbers of agents $n$. The Kripke model $M_n$ is succinctly represented by the succinct Kripke model $\mathfrak{M} = (AP_{M_n}, \beta_{M_n}, (\pi_a)_{a \in \text{AGT}})$ defined by $\beta_{M_n} = \top$, $AP_{M_n} = \{m_a, h_a | a \in \text{AGT}\}$ and $\pi_a = \text{assign}(\{m_a\} \cup \{h_b | b \neq a\})$.

The formula $\beta_{M_n} = \top$ means that the set of possible worlds is the set of all valuations. Program $\pi_a$ changes propositions agent $a$ is uncertain of ($m_a$ and $h_b$ for all $b \neq a$) while the truth values of other propositions remain unchanged. Pointed Kripke models $M, \mathfrak{M}$ and $\mathcal{M}(\mathfrak{M}), \mathfrak{w}$ are bisimilar. The number of worlds in $M_n$ is $2^{2n}$ while the size of $\mathfrak{M}$ is $O(n^2)$ (each program $\pi_a$ is of size $O(n)$).

Furthermore, there is no bisimilar Kripke model $M'$ with fewer worlds than $M_n$ since all valuations appear exactly once in $M_n$.

3.3 Succinct event models

We adopt a similar method for succinct event models.

3.3.1 Definition

Definition 18 A succinct event model is a tuple $E = (AP_E, \chi_E, (\pi_{a,E})_{a \in \text{AGT}}, \text{pre}, \text{post})$ where $AP_E$ is a set of atomic propositions disjoint from $AP$; $\chi_E$ is a propositional formula over $AP_E$ characterizing the set of events; $\pi_{a,E}$ is a program over $AP_E$ for all $a \in \text{AGT}$; $\text{pre}$ is a propositional formula over $AP_E \cup \mathcal{L}_{EL}(AP)$ (meaning that any atom from $AP_E$ cannot be under the scope of a $K$ or a $C$ operator); For all $p \in AP$, $\text{post}(p)$ is a propositional formula over $AP_E \cup \mathcal{L}_{EL}(AP)$.

3.3.2 From succinct event models to event models

Definition 19 Given a succinct event model $E = (AP_E, \chi_E, (\pi_{a,E})_{a \in \text{AGT}}, \text{pre}, \text{post})$, the event model represented by $E$, noted $\hat{E}(E)$ is the model $(E, (R^E_\text{a})_{a \in \text{AGT}}, \text{pre}, \text{post})$ on $AP$ where $E = \{v_e \in \nu(AP_E)| \nu_e \models \chi\}; R^E_\text{a} = \{(v_e, v_e') | v_e \overset{\pi_{a,E}}{\rightarrow} v_e'\}; \text{pre}(v_e) = \text{pre} \land \text{desc}(v_e); \text{post}(v_e, p) = \text{post}(p) \land \text{desc}(v_e)$. 
3.3. SUCCINCT EVENT MODELS

3.3.3 From event models to succinct event models

We define a succinct event model $\mathcal{E}_E$ representing the event model $\mathcal{E}$.

**Definition 20** Let $\mathcal{E} = (E, (R^e_a)_{a \in AGT}, pre, post)$ be an event model on $AP$. We define the succinct event model $\mathcal{E}_E$ as

$$\mathcal{E}_E = (AP_E, \chi_E, (\pi_{a,E})_{a \in AGT}, pre, post)$$

where $AP_E = \{p_e | e \in E\}$; $\chi_E = \exists! (AP_E)$; $\pi_{a,E} = \bigcup_{e \in R^e_a} p_e$; $pre = \bigwedge_{e \in E} (p_e \rightarrow pre(e))$; $post = \bigwedge_{e \in E} (p_e \rightarrow post(e))$.

The fresh atomic proposition $p_e$ designates event $e$. Formula $\chi_E$ describes the set $E$ and the precondition $pre$. Program $\pi_{a,E}$ performs a non-deterministic choice in the same spirit of $\pi_a$ in Definition 17. Program $post$ non-deterministically chooses the current event $e$ and applies the postcondition assignments in parallel. The following proposition states that $\mathcal{E}_E$ indeed represents $\mathcal{E}$.

**Example 29** The event model $\mathcal{E}$ of Figure 3.2 is modeled by the succinct event model $\mathcal{E}_E$ with $AP_E = \{p_e, p_f\}$, $\chi_E = \exists! (AP_E)$, $\pi_{a,E} = \bigcup_{e \in R^e_a} p_e$; $pre = \bigwedge_{e \in E} (p_e \rightarrow pre(e))$; $post = \bigwedge_{e \in E} (p_e \rightarrow post(e))$ (trivial postcondition counterpart in post has been omitted).

**Proposition 2** $\hat{E}(\mathcal{E}_E)$ and $\mathcal{E}$ are equivalent.

3.3.4 Succinctness of succinct event models

As for succinct Kripke models, we provide a family of event models having exponentially more compact representations with succinct event models.

**Example 30** We focus on the notion of attention-based announcement of $p$ as shown in [BeDH16]. In addition to classic atomic propositions, we add propositions $h_a$ for “agent $a$ is listening to the announcement”. The attention-based announcement of $p$ can then be represented by the event model $\mathcal{E} = (E, (\rightarrow^\xi_a)_{a \in AGT}, pre)$ where:

- $E = V(\{p\} \cup \{h_a \mid a \in AGT\}) \cup \{idle\}$
- for all $a$, $e \rightarrow^\xi_a f$ if $e \neq idle$, $f \neq idle$, $e \models h_a$ and $f \models p$
- for all $a$, $e \rightarrow^\xi idle$ if $e \neq idle$ and $e \not\models h_a$
- for all $a$, $idle \rightarrow^\xi idle$
- $pre(e) = \begin{cases} desc(e) & \text{if } e \neq idle \\ \top & \text{otherwise.} \end{cases}$
Figure 3.3 shows the event model of the attention-based announcement of two agents $a_1$ and $a_2$. The model consists of nine events. The six edges pointing to the dashed box point to all four events in the box.

Let us consider the family of models $\mathcal{E}_n$ given in Example 30 for all number of agents $n$. The event model $\mathcal{E}$ is succinctly represented by the succinct event model $\mathcal{E} = (\text{AP}_M, \text{AP}_E, \chi_E, (\pi_a, \chi_E, (\text{assign}(\{h_b, b \in AGT\}))) \cup (\neg h_a; \text{assign}(\text{AP}_M); p_{idle} \leftarrow \top)) \cup (p_{idle} = \top; \text{assign}(\text{AP}_M));$

- $\chi_E = \top;$
- $\pi_{a,E} = (\neg p_{idle}; (h_a; p \leftarrow \top; \text{assign}(\{h_b, b \in AGT\}))) \cup (\neg h_a; \text{assign}(\text{AP}_M); p_{idle} \leftarrow \top)) \cup (p_{idle} = \top; \text{assign}(\text{AP}_M));$
- $\text{post} = \top.$

Atomic proposition $p_{idle}$ intuitively means that the event $idle$ is occurring (at the bottom in Figure 3.3). Formula $\chi_E = \top$ means that the set of possible events is unconstrained. Program $\pi_{a,E}$ works as follows: if $p_{idle}$ is false, if $h_a$ is true, assign $\top$ to $p$ and arbitrarily change $h_b$ for all $b \neq a$; if $h_a$ is false, change valuations of propositions in $\text{AP}_M$ and set $p_{idle}$ to true; otherwise if $p_{idle}$ is true, change all truth values of propositions in $\text{AP}_M$. The number of worlds in $\mathcal{E}$ is $2^{n+1} + 1$ while the size of $\mathcal{E}$ is $O(n^2)$ (each program $\pi_{a,E}$ is of size $O(n)$).

Now we prove that standard event models cannot represent $\mathcal{E}_n$ as succinctly as succinct event models.

**Theorem 1** There is no propositional event model $\mathcal{E}'_n$ equivalent to $\mathcal{E}_n$ with fewer than $2^n$ events.
3.4. LANGUAGE OF SUCCINCT DEL

Proof.
We use the characterization of equivalent models with action emulations (Definition 12). We suppose that there is an action emulation $AE$ between $(E_n, e)$ and $(E'_n, e')$ for some $e \in E_n$ and $e' \in E'_n$. Let $\Sigma$ be the set of preconditions of $E_n$ and $E'_n$. Note that $\Sigma$ (defined as in Definition 12) is a set of propositional formulas. Let $e_1$ and $e_2$ be two valuations in $\mathcal{V}([p] \cup \{h_a \mid a \in AGT\})$ such that $e_1 \models p$ and $e_2 \models p$ and $e_1 \neq e_2$.

Suppose that there is an event $e'$ of $E'_n$, and $\Gamma_1, \Gamma_2$ such that $e_1AEr_1 e'$ and $e_2AEr_2 e'$. As $e_1 \neq e_2$, there is an agent $a$ such that $e_1 \models \neg h_a$ and $e_2 \models h_a$ (we swap $e_1$ and $e_2$ if $e_1 \models h_a$ and $e_2 \models \neg h_a$). Then $e_1 \rightarrow_{\Sigma} \text{idle}$. We consider the maximal consistent subset $\Gamma' = \{ \varphi \in \hat{\Sigma} \mid \{h_a, a \in AGT\} \models \varphi \}$. We have $\text{pre}(\text{idle}) \in \Gamma'$ and the formula $\left( \land_{\psi \in \Gamma_1} \psi \land \hat{K}_a \land_{\psi' \in \Gamma'} \psi' \right)$ is consistent (because $\Gamma_1$ and $\Gamma'$ are propositional). By Zig, there exists $f' \in E'$ such that $e' \rightarrow_{\Sigma} f'$ with $\text{idle}AE r_1 f'$. By Zag, as $e_2AEr_2 e'$ and the formula $\left( \land_{\psi \in \Gamma_2} \psi \land \hat{K}_a \land_{\psi' \in \Gamma'} \psi' \right)$ is consistent, there exists $f' \in E$ such that $e \rightarrow_{\Sigma} f$ and $fAE r_1 f'$. By invariance we obtain $\text{pre}(f') \in \Gamma'$. However, $\text{pre}(f) = \text{desc}(f)$ and $p \in f$ so $\{h_a, a \in AGT\} \models \text{pre}(f)$. Therefore we derive a contradiction.

This proves that there are at least $2^n$ events. ■

3.4 Language of succinct DEL

We define the language $\mathcal{L}_{\text{succDEL}}$:

$$\varphi ::= \top \mid p \mid \neg \varphi \mid (\varphi \lor \varphi) \mid K_a \varphi \mid \langle \mathcal{E}, \beta_0 \rangle \varphi$$

where $\mathcal{E}$ is a succinct event model over $(AP_M, AP_E)$ and $\beta_0$ is a Boolean formula over $AP_M \cup AP_E$.

The syntax of succinct DEL is similar to the syntax of DEL itself except that operators $\langle \mathcal{E}, \mathcal{E}_0 \rangle$ (where $\mathcal{E}, \mathcal{E}_0$ is a multi-pointed event model) are replaced by $\langle \mathcal{E}, \beta_0 \rangle$ where $\beta_0$ is a Boolean formula that succinctly represents a set of events $\mathcal{E}_0$. The semantics of $\langle \mathcal{E}, \beta_0 \rangle \varphi$ is:

• $\mathcal{M}, w \models \langle \mathcal{E}, \beta_0 \rangle \varphi$ iff there exists $e \in \hat{\mathcal{E}}(\mathcal{E})$ such that $e \models \beta_0$, $\mathcal{M}, w \models \text{pre}(e)$ and $\mathcal{M} \odot \hat{\mathcal{E}}(\mathcal{E}), (w, e) \models \varphi$ where $\text{pre}(e)$ is the precondition of $e$ in $\hat{\mathcal{E}}(\mathcal{E})$.

We write $\mathcal{M} \odot \hat{\mathcal{E}}, w \models \varphi$ for $\hat{\mathcal{M}}(\mathcal{M} \odot \hat{\mathcal{E}}), w \models \varphi$. 

Part II

Decision problems
Chapter 4

Model checking

In this chapter, we will discuss the following model checking problem.

Definition 21 The model checking of DEL is defined as follows:

- Input: a pointed epistemic model $M, w$ and a formula $\varphi \in \mathcal{L}_{DELCK}$;
- Output: yes if $M, w \models \varphi$, no otherwise.

4.1 Motivation

In 1991, Halpern and Vardi wrote a manifesto published at KR [HV91], in favor of using model checking in applications where agents have knowledge, as opposed to using theorem proving. Typically, in model checking, a pointed epistemic model completely describes the current state of affairs and is given as an input, together with a formula to check.

For instance, let us consider a centralized application: offline video games. In an offline video game, all the data are stored in a single computer and the real world $w$ is stored. A pointed epistemic model $M, w$ may represent the epistemic situation of all virtual agents in a given scene. Virtual agents may be programmed with knowledge-based programs where conditions in if-statements are epistemic properties. We then require to evaluate the truth value of a given if-statement condition $\varphi$ in the current state of affairs $M, w$.

In a decentralized application such as a system of autonomous robots that do not share memory, we may imagine that each robot $a$ is equipped with an epistemic model $M$ and a set of possible worlds she considers as possible (see [Auc10]). In that case, each robot may evaluate formulas of the form $K_a \varphi$, whose semantics does not depend on the chosen possible world she considers as possible. Therefore, in the decentralized case, we can still use a model checker that solves the model checking problem of Definition 21.

4.2 A first algorithm

Figure 4.1 shows an algorithm that takes a model $M$ and property $\varphi$ and returns the set of $\varphi$-worlds. The presentation is equivalent to algorithms with marking [BS99].

However, the bottleneck is the computations of the product $M \otimes E$ leading to epistemic models of exponential size in the size of the input. However, in practice, this form is suitable for an implementation since sets of worlds can be represented efficiently with BDDs ([Bou76]).
CHAPTER 4. MODEL CHECKING

function mc(M, ϕ)
match ϕ do
  case p :
    return {w | p holds in M, w}
  case ¬ψ :
    return mc(M, ψ)
  case (ψ1 ∨ ψ2) :
    return mc(M, ψ1) ∪ mc(M, ψ2)
  case Kaψ :
    return {w | Ra(w) ⊆ mc(M, ψ)}
  case ⟨E, e⟩ψ :
    return mc(M, pre(e)) ∩ {w | (w, f) ∈ mc(M ⊗ E, ψ)}
endFunction

Figure 4.1: Model checking algorithm.

[Ake78], [Lee59]). It has recently been applied to DEL [vBvEGS15a]. We will see in Section 4.3 an algorithm for model checking that runs in polynomial space and that never computes products explicitly.

However, the algorithm given in Figure 4.1 is suitable in terms of complexity for \( L_{EL} \) (epistemic logic without actions) and for public announcement logic.

The model checking of \( L_{EL} \) is P-complete. The P-hardness proof can be found in [Sch02].

The P-membership is folklore: the algorithm in Figure 4.1 without the \( ⟨E, e⟩ψ \)-case can be implemented in polynomial-time.

In public announcement logic, all event models \( E \) are reflexive single-event models, therefore all models in the subcall in the algorithm 4.1 are submodels of the initial model \( M \). Therefore all operators performed in a node of the computer tree of \( mc(M, ϕ) \) is polynomial in the size of the input. To sum up, the model checking of public announcement logic is also in P [Ben11].

4.3 PSPACE-membership

In this section, we prove the following theorem, given in [CS18], and that subsumes the upper bound given in [AS13] for the model checking problem for \( L_{DEL} \).

Theorem 2 The model checking problem for DELCK is PSPACE-complete.

Hardness comes directly from the PSPACE-hardness of the model checking of DEL without common knowledge [AS13] (actually, it is already PSPACE-hard for single-pointed event models [BJS15a], but actually even when the Kripke model is S5 and event models are S5 and single-pointed [vdPvRS15]). For the PSPACE-membership, Figure 4.3 provides the pseudo-code of an alternating Turing machine that decides the model checking problem for DELCK in polynomial time. In the pseudo-code in Figure 4.3 existential (∃) and keyword or (resp. universal (∀) choices and keyword and) corresponds to existential (resp. universal) states. Keyword not corresponds to a negated state. The upper bound is proven since PSPACE = APTIME. The machine starts by calling \( mc(M, w, Φ) \). Figure 4.2 gives an example: it shows a portion of the computation tree of the call \( mc(M, w, ⟨E, E_0⟩¬C_{GP}) \).

Epistemic logic is an extension of the fragment of CTL with only the next operators AX and EX, proven to be P-hard.
4.3. PSPACE-MEMBERSHIP

Figure 4.2: Computation tree rooted at $mc(M, w, (E, E_0) \neg CGp)$.

The specifications of the procedures $mc$, $inval$ in, $rel$ and $rel^*$ (see Figure 4.3) are given in the following proposition. Note that the beginning of a procedure call is indeed a special configuration of the alternating Turing machine of Figure 4.3.

**Proposition 3** For all formulas $\varphi$, for all Kripke models $M$, for all sequences of event models $\overrightarrow{E}$, for all worlds $w \overrightarrow{e}$, for all states $a$, for all groups of agents $G$, for integers $i$ that are powers of two,

- configuration $mc(M_\overrightarrow{E}, w \overrightarrow{e}, \varphi)$ is accepting iff $M_\overrightarrow{E}, w \overrightarrow{e} \models \varphi$,
- configuration $inval(p, w \overrightarrow{e}, M_\overrightarrow{E})$ is accepting iff $p \in V(w \overrightarrow{e})$,
- configuration $in(u \overrightarrow{f}, M_\overrightarrow{E})$ is accepting iff $u \overrightarrow{f} \in M_\overrightarrow{E}$,
- configuration $rel(u \overrightarrow{f}, u \overrightarrow{f}, a, M_\overrightarrow{E})$ is accepting iff $(u \overrightarrow{e}, u \overrightarrow{f}) \in R_a$,
- and configuration $rel^*(u \overrightarrow{e}, u \overrightarrow{f}, G, i, M_\overrightarrow{E})$ is accepting iff $(u \overrightarrow{e}, u \overrightarrow{f}) \in \bigcup_{j \leq i} \left( \bigcup_{a \in G} R_a \right)^j$.

**Proof.**

The proposition is straightforwardly proven by induction since the pseudo-code directly reflects the semantics of DELCK. The only difficulties are:

- The induction works on the quantities given in Figure 4.3 and thanks to the following Lemma

  \[ \bigcup_{j \leq B_{M, \Phi}} \left( \bigcup_{a \in G} R_a \right)^j = \left( \bigcup_{a \in G} R_a \right)^* \]

- The design of Procedure $rel^*$ relies on the divide and conquer paradigm. For checking that $u \overrightarrow{f}$ is reachable by at most $i \bigcup_{a \in G} R_a$-steps from $w \overrightarrow{e}$, we guess an intermediate
Figure 4.3: Model checking procedures for DELCK (in gray: quantities associated to each procedure call).

Rather than giving a tedious and straightforward proof of Proposition 3, let us explain the algorithm on the example of \( mc(M, w, (E, E_0) \neg C_{GP}) \) depicted in Figure 4.2. The procedure
starts by choosing $e$ in $E_0$. Then, we check that both $pre(e)$ holds in $w$ and that $\neg C_{GP}$ holds in $we$. Checking that $\neg C_{GP}$ holds in $we$ leads to a negated configuration: we negate the fact that $C_{GP}$ holds in $we$. It is followed by a universal choice of $uf \in M\mathcal{E}$. For each choice of $uf \in M\mathcal{E}$, we progress in an existent configuration that checks that either $uf$ is not a world of $M\mathcal{E}$, or $uf$ is not reachable from $we$ by at most $R^u_{B,M,\Phi}$ steps or that $p$ in $uf$. Checking that $p$ holds in $uf$ is performed by the call of $\text{invalid}(p, uf, M\mathcal{E})$, which itself checks that the postcondition $post(e, p)$ holds in $u$.

The quantities associated to each procedure call used for the proofs by induction require a careful definition. They depend on the input $(M, w, \Phi)$ of the model checking problem. Let $B_{M,\Phi}$ be the smallest power of two that is greater than the number of worlds in Kripke model $M\mathcal{E}$ where $\mathcal{E}$ is the list of all event models appearing in the formula $\Phi$.

**Definition 22** $|M|$, $|\mathcal{E}|$ and $|\varphi|$ are defined by mutual induction. First, $|M|$ (resp. $|\mathcal{E}|$) are the number of bits to encode Kripke model $M$ (resp. event model $\mathcal{E}$). In particular, $|\mathcal{E}|$ takes into account the memory needed to store the precondition and the postcondition functions. Then, $|M\mathcal{E}|$ denotes $|M| + \sum_{i=1}^{n} |\mathcal{E}_i|$. Finally $|\varphi|$ denotes the length of $\varphi$, defined by induction as usual except for the two following cases:

- $|(\mathcal{E}, E_0)\varphi| := |\mathcal{E}| + 1 + |\varphi|$;
- $|C_G\varphi| := \log_2 B_{M,\Phi} + 1 + |\varphi|$.

**Lemma 1** The quantities given in gray in Figure 4.3 are strictly decreasing along a branch of the computation tree (see for instance Figure 4.2 of $mc(M, w, \Phi)$.

**Proof.**

Let us discuss the following cases (the other ones are left to the reader):

- The quantity for $mc(M\mathcal{E}, w, C_G\varphi)$ is $|M\mathcal{E}| + |C_G\varphi| + 1 = |M\mathcal{E}| + \log_2 B_{M,\Phi} + |\varphi| + 1$ and is strictly greater than the quantity for $rel^*(w^e, u^f, G, B_{M,\Phi}, M\mathcal{E})$, which is $|M\mathcal{E}| + \log_2 B_{M,\Phi}$.
- The quantity for $mc(M\mathcal{E}, w, (\mathcal{E}, E_0)\varphi)$ is $|M\mathcal{E}| + |\mathcal{E}| + |\varphi| + 1$ and is strictly greater than the quantity for $mc(M\mathcal{E}, w, E^e, pre(e))$, which is $|M\mathcal{E}| + |pre(e)| < |M\mathcal{E}| + |\mathcal{E}|$.
- The quantity for $\text{invalid}(p, w, M\mathcal{E})$ is $|M\mathcal{E}| + |\mathcal{E}|$ and is strictly greater than the quantity for $mc(M\mathcal{E}', w, post(e, p))$ which $|M\mathcal{E}'| + |post(e, p)|$.

**Proposition 4** $mc(M, w, \Phi)$ is executed in polynomial time in the size of the input $(M, w, \Phi)$.

**Proof.**

The time is bounded by the height of the computation tree rooted in $mc(M, w, \Phi)$. Thanks to Lemma 1, the height of the computation tree (as shown in Figure 4.2) is bounded by the quantity associated to $mc(M\mathcal{E}, w, \varphi)$, that is $|M| + |\varphi|$. This quantity is not the size of the input $(M, w, \Phi)$: for instance the weight in this quantity of $C_G$-modalities is $\log_2 B_{M,\Phi}$. However this quantity is polynomial in the size of the input $(M, w, \Phi)$.

At each node of the computation tree (as depicted in Figure 4.2), the computation performed in a single node is polynomial. For instance, the instruction (V) choose $uf \in M\mathcal{E}'$ consists of choosing each bit of $u^f$, thus is polynomial in the size of the input.

To conclude, the execution time on each branch in the computation tree is polynomial.
CHAPTER 4. MODEL CHECKING

### 4.4 Other results

In this section, we discuss these results and other more anecdotic results shown in the literature.

#### 4.4.1 Explicit models

Aucher et al. [AS13] prove that the model checking problem with the $\cup$ operator is PSPACE-hard. Van de Pol et al. (see Theorem 1 in [vdPvRS15]) and Bolander et al. (see Theorem 6.1 in [BJS15b]) independently prove that the model checking problem is PSPACE-hard even without the $\cup$-operator.

The complexity of model checking of attention-based public announcement logic is not given in [BvDH +16]: it is in PSPACE but the exact complexity is unknown. In the special case of no propositional preconditions and no postconditions, action models commute [LPW11]. Charrier et al. exploit this property to show that the model checking of formulas containing operators of the $\langle (E, e)^i \rangle$ where $i$ is an integer written in binary is in PSPACE (Proposition 1 in [CMS16]). Their algorithm is a deterministic algorithm that runs in polynomial space. A tuple $((w, e_1, \ldots, e_k))$ is represented by $((w, j_e)_{e \in E})$ where $E$ is the set of events and $j_e$ is the number of occurrences of $e$ in the tuple $e_1, \ldots, e_k$. The space required to represent $((w, j_e)_{e \in E})$ is polynomial in the size of the input, therefore the algorithm runs in polynomial space.

The model checking problems of both arbitrary public announcement logic and group announcement logic are PSPACE-complete [˚ABvDS10].

#### 4.4.2 Succinct models

When the Kripke model is described succinctly, the model checking of formulas without dynamic operators is already PSPACE-hard [BHT13].

When the Kripke model and event models are described succinctly by means of accessibility programs, the model checking for $\mathcal{L}_{DELCK}$ is in PSPACE ([CS17], [CS18]). The idea is to use a similar algorithm to the one of Figure 4.3 but that manipulates succinct models instead.

Actually, when the Kripke model is succinctly described by means of accessibility programs, the model checking for arbitrary public announcement is $A_{pol}$EXPTIME-complete [CS15]. The $A_{pol}$EXPTIME-upper bound is shown as follows. The main difference in the algorithm is that we now store the subset of surviving worlds explicitly as a set of valuations. The time of an operation performed in a node is exponential. Checking that $\Box \psi$ holds at a given world consists of searching for the existence of a subset of valuations being the next surviving set of worlds for which $\psi$ holds. We model it as an exponential number of atomic non-deterministic existential choices. Dually, checking that $\neg \Box \psi$ holds leads to an exponential number of atomic universal choices. As the depth of the tree is linear the size of $\varphi$, the number of alternation over a branch of the computation tree is linear in the size of $\varphi$. 

---

| $\mathcal{L}_{EL, L_{ELCK}}$ | P-complete | PSPACE-complete |
| $\mathcal{L}_{ELPA, L_{ELCKPA}}$ | P-complete | PSPACE-complete |
| $\mathcal{L}_{DEL, L_{DELCK}}$ | PSPACE-complete | PSPACE-complete |
| with arbitrary public announcements | PSPACE-complete | $A_{pol}$EXPTIME-complete |

Table 4.1: Complexity of the model checking problem.
The main complexity results for model checking are summed up in Table 4.1 and give an insight into the implementability of the model checking of DEL. Model checking on explicit models is implemented in *Hintikka’s world*. Model checking on succinct models is implemented in a tool called DEMO ([vBvEGS15a], [vBvEGS18]). Model checking on succinct models for arbitrary public announcement logic is implemented by reduction to a first-order prover *iprover* [CPS17].
Chapter 5

Satisfiability problem

In this chapter, we will discuss the following satisfiability problem.

**Definition 23** The satisfiability problem of DEL is defined as follows:

- **Input:** a formula $\varphi \in L_{\text{DELCK}}$
- **Output:** yes if there exists a pointed epistemic model $M, w$ such that $M, w \models \varphi$, no otherwise.

5.1 Motivation

The model checking methodology relies on the fact the full Kripke model of the epistemic state has to be described. As pointed by Sergei Artemov [Art16], a full description is problematic for mainly two reasons.

1. From an applicative point of view, we often actually do not have a complete description of a situation, but a partial description.
2. From a conceptual point of view, having a single Kripke model means that this very model is common knowledge among all agents.

That is why, in this chapter, we focus on the satisfiability problem. It can be used to generate an epistemic state that respects a given property $\varphi$ holds. Hintikka’s World enables to perform this task (build a custom example). Also the satisfiability problem can be seen as the dual problem of validity checking / theorem proving. As an application, we can check a property $\psi$ in a system described textually with $\chi$. The idea is then to check that the formula $\chi \rightarrow \psi$ is valid $(\chi \land \neg \psi)$ is unsatisfiable.

5.2 Tableau method

5.2.1 Principle

The general idea is to try to analyze a given formula and to create an epistemic model $(M, w)$ that satisfies that formula. Let $\mathbb{L}ab$ be a countable set of labels designed to represent possible worlds of the epistemic model $(M, w)$ in creation. Our tableau method manipulates terms that we call tableau terms and they are of the following kind:
• $(\sigma \vec{e} \varphi)$ where $\sigma \in \text{Lab}$ is a node (that represents a world in the epistemic model) and $\vec{e}$ is a finite sequence of events (without loss of generality, we suppose that all dynamic operators refer to the same event model $\mathcal{E}$ so it is not necessary to mention it). This term means that formula $\varphi$ is true in the world $\sigma \vec{e}$, that is, the world obtained from the world denoted by $\sigma$ after executing the sequence $\vec{e}$ – it also implicitly implies that the sequence $\vec{e}$ is executable in the world denoted by $\sigma$;

• $(\sigma \vec{e} \checkmark)$ means that the sequence $\vec{e}$ is executable in the world denoted by $\sigma$;

• $(\sigma \vec{e} \otimes)$ means that the sequence $\vec{e}$ is not executable in the world denoted by $\sigma$;

• $(\sigma R_a \sigma_1)$ means that the world denoted by $\sigma$ is linked by $R_a$ to the world denoted by $\sigma_1$;

• $\bot$ denotes an inconsistency.

A tableau rule is represented by a numerator $\mathcal{N}$ above a line and a finite list of denominators $\mathcal{D}_1, \ldots, \mathcal{D}_k$ below this line, separated by vertical bars:

$$
\begin{array}{c}
\mathcal{N} \\
\mathcal{D}_1 | \ldots | \mathcal{D}_k
\end{array}
$$

The numerator and the denominators are finite sets of tableau terms.

A tableau tree is a finite tree with a set of tableau terms at each node. A rule with numerator $\mathcal{N}$ and denominator $\mathcal{D}$ is applicable to a node carrying a set $\Gamma$ if $\Gamma$ contains an instance of $\mathcal{N}$ but not the instance of its denominator $\mathcal{D}$. If no rule is applicable, $\Gamma$ is said to be saturated. We call a node $\sigma$ an end node if the set of formulas $\Gamma$ it carries is saturated, or if $\bot \in \Gamma$. The tableau tree is extended as follows:

1. Choose a leaf node $n$ carrying $\Gamma$ where $n$ is not an end node, and choose a rule $\rho$ applicable to $n$.

2. (a) If $\rho$ has only one denominator, add the appropriate instantiation to $\Gamma$.

   (b) If $\rho$ has multiple denominators, choose one of them and add to $\Gamma$ the appropriate instantiation of this denominator.

A branch in a tableau tree is a path from the root to an end node. A branch is closed if its end node contains $\bot$, otherwise it is open. A tableau tree is closed if all its branches are closed, otherwise it is open. The tableau tree for a formula $\varphi$ is the tableau tree obtained from the root $\{(\sigma_0 \epsilon \varphi)\}$ when all leafs are end nodes.

5.2.2 Tableau rules

The rules of our tableau method are represented in Figure 5.1. In these rules, $\epsilon$ is the empty sequence of events. The tableau method contains the classical Boolean rules $(\land), (\neg \neg), (\neg \land)$. The rules $\left(\leftarrow \mu\right)$ and $\left(\leftarrow \neg p\right)$ handle atomic propositions. The rule $\left(\bot\right)$ makes the current execution fail. The rule for $(K_a)$ is applied if $\vec{e}$ and $\vec{f}$ are such that the lengths of $\vec{e}$ and $\vec{f}$ are equal and for all $j$, $e_j = f_j$. Similarly, the rule for $(\hat{K}_a)$ is applied by choosing non-deterministically some $\vec{f}$ of the same length than $\vec{e}$ such that $e_j R_a f_j$ and creating a new fresh label $\sigma_{\text{new}}$. The rules $(\checkmark), (\otimes), (\text{clash}_{\checkmark, \otimes})$ and $(\epsilon_{\otimes})$ handle the preconditions.

Rule $(\left[\epsilon\right])$ is the rule for the dynamic operator for single-pointed event models. It says that if $[\epsilon]\varphi$ holds in $\sigma$ after the sequence of actions $\vec{e}$, then:

• either event $e$ is not executable in $\sigma$;
• or event e is executable in σ and ϕ holds after the sequence $e^\triangleright e$ in σ.

Rules $(E_0 \cup \{e\})$ and $(|E_0 \cup \{e\}|)$ decompose the multi-point set of multi-point event models. Rule $(|E_0 \cup \{e\}|)$ says that both $|E_0|$ and $|e|$ holds in σ after $e^\triangleright e$. On the contrary, rule $(|E_0 \cup \{e\}|)$ is non-deterministic:

• Either $(E_0)$ holds in σ after $e^\triangleright$ (one of the events in $E_0$ is executable);

• or $\langle e \rangle$ holds in σ after $e^\triangleright$.

Rule $(\langle \emptyset \rangle)$ says that any multi-point event models with an empty designated set of events is not executable.

### 5.3 The common knowledge free fragment

The satisfiability problem for the language $\mathcal{L}_{\text{DEL}}$ is NEXPTIME-complete and the tableau method given in Figure 5.1 is optimal [AS13].

The NEXPTIME upper bound is explained as follows. First, the tableau method is a non-deterministic algorithm. The depth of the computation tree corresponding to the tableau is polynomial in the input ϕ but the number of tableau terms corresponding to a given a node $\sigma \in \text{Lab}$ is at most exponential in the input ϕ. So the non-deterministic algorithm runs in exponential time.
The NEXPTIME-lower bound is proven by reduction from a tiling problem but needs the \( \cup \)-operator [AS13]. We do not know whether the satisfiability problem for the language without the \( \cup \)-operator is NEXPTIME-hard.

The satisfiability problem of public announcement logic is PSPACE-complete [Lut06]. An optimal tableau method is given in [BvDHL10] and can be seen as a restriction to the tableau method given in Section 5.2: the possible sequences \( \vec{e} \) correspond to sequences of nested announcements.

Example 31 By checking the satisfiability of \( \varphi(p \wedge [\psi]q) \wedge [\chi]r \), the set of possible sequences \( \vec{e} \) are \( \epsilon \), \([\epsilon_\varphi]\), \([\epsilon_\varphi, \epsilon_\psi]\) and \([\epsilon_\chi]\) where \( \epsilon_\varphi \), \( \epsilon_\psi \) and \( \epsilon_\chi \) are respectively the events corresponding to the public announcements of \( \varphi \), \( \psi \) and \( \chi \).

The satisfiability problem of attention-based announcement logic is also PSPACE-complete [BvDH+16]. The satisfiability problem of non-normal modal logic variants of public announcement logic have been proven to be NP-complete [MSSV15].

5.4 Common knowledge

The complexity of the satisfiability of \( L_{ELCK} \) is EXPTIME-complete. It remains EXPTIME-complete when extended with public announcement operators [Lut06]. The idea is to reduce the satisfiability with relativized common knowledge.

Reduction axioms for common knowledge that use relativized common knowledge also exist for event models [WA15]: we can transform a formula containing pointed event models in a formula without any occurrence of a pointed event model but that contains relativized common knowledge. The translation is non-elementary. Actually, the tableau method can be extended by the following rules [CS18]:

\[
\begin{align*}
\frac{\sigma \, \vec{e} \, C_G \varphi}{\sigma \, \vec{e} \, \varphi} & \quad \frac{\sigma \, \vec{e} \, C_G \varphi}{\sigma \, \vec{e} \, \varphi} \\
\frac{\sigma_1 \, \vec{f} \, \top}{\sigma_1 \, \vec{f} \, \varphi} & \quad \frac{\sigma_1 \, \vec{f} \, \top}{\sigma_1 \, \vec{f} \, \varphi} \\
\frac{\sigma \, \vec{e} \, \hat{C}_G \varphi}{\sigma \, \vec{e} \, \varphi} & \quad \frac{\sigma \, \vec{e} \, \hat{C}_G \varphi}{\sigma \, \vec{e} \, \varphi}
\end{align*}
\]

5.4.1 Upper bound

The satisfiability problem for DELCK, given a DELCK-formula \( \Phi \), asks to decide \( \Phi \) is satisfiable. In this section, we prove the following upper bound result:

Theorem 3 The satisfiability problem of DELCK is in 2-EXPTIME.
5.4. COMMON KNOWLEDGE

Definition 24 The closure\(^1\) of formula \(\Phi\) is the set \(\text{Cl}(\Phi)\) that contains elements in\(\mathfrak{P}\) and \((\overline{e} \varphi)\) where \(\overline{e}\) is a sequence of events in \(E\) of length at most \(\text{dmd}(\Phi)\), and \(\varphi\) is a subformula (or negation) of \(\Phi\) or a subformula (or negation) of a precondition or postcondition in \(E\), under the condition that \(\text{dmd}(\varphi) + |\overline{e}| \leq \text{dmd}(\Phi)\).

The intended meaning of in\(\mathfrak{P}\) is that the current world survives the sequence of events \(\overline{e}\). The intended meaning of \((\overline{e} \varphi)\) is that formula \(\varphi\) is true after having executed the sequence of events \(\overline{e}\).

Example 32 Let us take the event model \(\mathcal{E}\) of Figure 2.7 and formula \(\Phi := [e]K_a[f]q\). The closure \(\text{Cl}(\Phi)\) is the set \(\{ in_e, in_e, in_e, in_{f}, in_{f}, (e \ \mathfrak{P} \ K_a[f]q) \} \).

Proposition 5 The size of the closure of \(\Phi\) is exponential in |\(\Phi\)|.

Proof. There is a direct correspondence between a subformula of \(\Phi\) and a node in the syntactic tree of \(\Phi\). Therefore, the number of subformulas of \(\Phi\) is in \(O(|\Phi|)\). The number of possible \(\psi\) is then bounded by \(O(|\Phi|)\) (the size of \(\Phi\) is the number of memory cells needed to write down \(\Phi\), all the information of the event model \(\mathcal{E}\) included). The number of possible sequences \(\overline{e}\) is \(|E|^{\text{dmd}(\Phi)}\), thus exponential in |\(\Phi\)|. ■

A Hintikka set (see Definition\(^2\)) is a maximal subset of \(\text{Cl}(\Phi)\) that is consistent with respect to propositional logic (points 2-4), common knowledge reflexivity (point 5), dynamic operators (point 6-7), executability of events (points 8-9) and postconditions (point 10).

Definition 25 A Hintikka set \(h\) over \(\text{Cl}(\Phi)\) is a subset of \(\text{Cl}(\Phi)\) that satisfies:

1. If \((\overline{e} \varphi) \in h\) then in\(\mathfrak{P}\) \(\in h\);
2. \((\overline{e} \varphi \wedge \psi) \in h\) iff \((\overline{e} \varphi) \in h\) and \((\overline{e} \psi) \in h\);
3. \((\overline{e} \varphi \vee \psi) \in h\) iff \((\overline{e} \varphi) \in h\) or \((\overline{e} \psi) \in h\);
4. If in\(\mathfrak{P}\) \(\in h\) then \((\overline{e} \varphi) \in h\) and \((\overline{e} \neg \varphi) \in h\);
5. If \((\overline{e} \mathcal{C}_G \varphi) \in h\) then \((\overline{e} \varphi) \in h\);
6. \((\overline{e} [E_0] \varphi) \in h\) iff there exists \(e \in E_0\) s.t. in\(\mathfrak{P}\) \(\in h\) and \((\overline{e} :: e \ \varphi) \in h\)\(^3\);
7. \((\overline{e} [E_0] \varphi) \in h\) iff for all \(e \in E_0\), we have in\(\mathfrak{P}\) \(\in h\) implies \((\overline{e} :: e \ \varphi) \in h\);
8. \(in_e \in h\);
9. \(in_{\mathcal{E},e} \in h\) iff \(in_{\mathfrak{P}} \in h\) and \((\overline{e} \ \mathfrak{P} \ e) \in h\);
10. \((\overline{e} :: e \ p) \in h\) iff \(\overline{e} \mathfrak{P} \text{ post}(e)(p) \in h\).

Point (1) means that a Hintikka set contains \((\overline{e} \varphi)\), then it means that \(\overline{e}\) should be executable (in the intuitive world represented by the Hintikka set). Point (4) means that Hintikka sets are consistent. Point (5) says that if \(\varphi\) is common knowledge then \(\varphi\) is true. Points (6) and (7) mimics the truth condition given in Definition 8. Point (8) means the empty sequence of events \(e\) is always executable. Point (9) means that \(\overline{e} :: e\) is executable iff \(\overline{e}\) is executable and the precondion of \(e\) holds after having executed \(\overline{e}\). Point (10) means that the truth of atomic proposition \(p\) after a non-empty sequence \(\overline{e} :: e\) of events is given by the truth of its postcondition before the last event \(e\).

Now, we define the following structure that takes care about the consistency of the box modalities \(K_a, C_G\).

\(^1\)The definition given here contains ‘too many’ formulas. We could have given a much more thorough definition, but the definition would have been more complicated to understand and the closure would have had the same asymptotic size.

\(^2\)Formula \(\neg \varphi\) is the negation of \(\varphi\) in the following sense: the negative normal obtained by negating all connectives in \(\varphi\).

\(^3\)We explicitly mentioned in\(\mathfrak{P}\) \(\in h\) for uniformity with the semantics. However, note that it is implied by point (1).
function \textsc{isDELCK-sat}?(\Phi)
Compute the Hintikka structure \( H := (H, (R_a)_{a \in AGT}) \) for \( \Phi \)
repeat
- Remove any Hintikka set \( h \) from \( H \) if
  - \( (K_a) \) either there is \((\vec{c}, K_a \psi) \in h \) but no \( h' \in R_a(h) \) with \( \vec{c} \rightarrow^a \vec{c}' \) and \( \text{in}_{\vec{c}'} \in h' \);
  - \( (\hat{C}_G) \) or there is \((\vec{c}, \hat{C}_G \psi) \in h \) but no path \( h = h_0 \rightarrow^{\alpha_1} h_1 \ldots h_k \) and no path \( \vec{c} = \vec{c}^{(0)} \rightarrow^{\alpha_1} \vec{c}^{(1)} \ldots \rightarrow^{\alpha_k} \vec{c}^{(k)} \) such that \((\vec{c}^{(k)} \psi) \in h_k \) and \( a_1, \ldots, a_k \in G \) and \( \text{in}_{\vec{c}^{(i)}} \in h_i \).
until no more Hintikka sets are removed
if there is still a Hintikka set in \( H \) containing \((e, \Phi) \) then accept else reject
endFunction

Figure 5.2: Algorithm for the satisfiability problem of a \textsc{DELCK}-formula \( \Phi \).

Definition 26 The Hintikka structure for \( \varphi \) is \( H := (H, (R_a)_{a \in AGT}) \) where:

- \( H \) is the set of all possible Hintikka sets over \( Cl(\Phi) \);
- \( hR_a h' \) if the two following conditions holds:
  - \( (K_a) \) for all \((\vec{c}, K_a \varphi) \in h \) we have \((\vec{c}', \varphi) \in h' \) for all \( \vec{c}' \) such that \( \vec{c} \rightarrow^a \vec{c}' \) and \( \text{in}_{\vec{c}'} \in h' \),
  - \( (\hat{C}_G) \) for all \((\vec{c}, \hat{C}_G \varphi) \in h \) we have \((\vec{c}', \hat{C}_G \varphi) \in h' \) for all \( \vec{c}' \) such that \( \vec{c} \rightarrow^a \vec{c}' \) with \( a \in G \) and \( \text{in}_{\vec{c}'} \in h' \).

The size of the Hintikka structure is double-exponential in \(|\varphi|\), since there are a double-exponential number of different Hintikka sets. We finish by giving the algorithm \textsc{isDELCK-sat}? (see Figure 5.2) whose repeat...until loop takes care about the consistency of diamond modalities, \( K_a, \hat{C}_a \). The algorithm starts with the full Hintikka structure. Points \((K_a), (\hat{C}_a)\) remove worlds where \( K_a \psi \) and \( \hat{C}_a \psi \) have no appropriate \( \psi \)-successor. We write \( \vec{c} \rightarrow^a \vec{c}' \) if for all \((\vec{c}', \vec{c}^{(i)}), \in \) \( R^{(i)}_a \). Actually, the algorithm decides in double-exponential time whether a \textsc{DELCK}-formula is satisfiable (Propositions 6 and 7).

Proposition 6 Algorithm \textsc{isDELCK-sat}? of Figure 5.2 runs in double-exponential time in \(|\Phi|\).

Proof.
The computation of \( H \) can be performed by brute-force: enumerate all subsets of \( Cl(\Phi) \) and discard those which does not satisfy all conditions (1)-(10) of Definition 26. Compute \( R_a \) according to Definition 26. The loop is repeated at most the number of Hintikka sets in \( H \), that is \( O(2^{2|\Phi|}) \) times, since at least one Hintikka set is removed or we exit the loop. Both tests \((K_a)\) and \((\hat{C}_G)\) can be performed by depth-first search algorithm running in polynomial time in the size of the graph, that is of size double-exponential in \(|\Phi|\).

Proposition 7 \( \Phi \) is \textsc{DELCK}-satisfiable iff \textsc{isDELCK-sat}? accepts \( \Phi \).

Proof.
\((\Rightarrow)\) Let \( M, w \) such that \( M, w \models \Phi \). Given a world \( u \), we note \( h(u) \) the Hintikka set obtained by taking \( \text{in}_{\vec{c}} \) if \( \vec{c} \) is executable in \( u \) and \((\vec{c}, \psi) \) if \( \psi \) holds in \( u, \vec{c} \). We show that no Hintikka
set $h(u)$ is removed from $\mathcal{H}$. In particular, $h(w)$ is not removed, and contains $(\epsilon \Phi)$ so the algorithm is $\textsc{DELC}\text{-}\text{sat}$? accepts $\Phi$.

$(\Leftarrow)$ Suppose $\textsc{DELC}\text{-}\text{sat}$? accepts $\Phi$. We construct a model $\mathcal{M} = (W, \langle R_a \rangle_{a \in \text{AGT}}, V)$ as follows:

- $W$ is the set of Hintikka sets that remain in the structure at the end of the algorithm;
- $R_a$ is the relation for agent $a$ at the end of the algorithm;
- $V(h) = \{ p \in AP \mid (\epsilon p) \in h \}$.

The proof finishes by proving the following lemma:

**Lemma 2** (truth lemma) The properties $\mathcal{P}(\text{in}_\mathcal{E})$ and $\mathcal{P}((\mathcal{E} \varphi))$ defined below hold:

- $\mathcal{P}(\text{in}_\mathcal{E})$: for all $h \in W$, $\text{in}_\mathcal{E} \in h$ iff $\mathcal{E}$ is executable in $\mathcal{M}, h$;
- $\mathcal{P}((\mathcal{E} \varphi))$: for all $h \in W$, $(\mathcal{E} \varphi) \in h$ iff $\mathcal{M} \otimes \mathcal{E}|_h, (h, \mathcal{E} \varphi) \models \varphi$.

**Proof.**

The proof is performed by induction by assigning the following quantities: the quantity for $\text{in}_\mathcal{E}$ is $n|\mathcal{E}|$; the quantity for $(\mathcal{E} \varphi)$ is $n|\mathcal{E}| + |\varphi|$ where $n$ is the length of $\mathcal{E}$, $|\mathcal{E}|$ and $|\varphi|$ are defined as in Definition 22 except that now we use the traditional clause $|C_G\varphi| := |\varphi| + 1$. Let us just present the following induction cases.

- Suppose that for all $e, e \in E_0$ s.t. $\mathcal{P}(\text{in}_\mathcal{E}, e)$ and $\mathcal{P}((\mathcal{E} : e \varphi))$. Let us prove that $\mathcal{P}((\mathcal{E} \langle E_0 \rangle \varphi))$. Let $h$ be a remaining Hintikka set.

  $(\Leftarrow)$ Suppose that $\mathcal{E} \varphi \in h$. It implies that for all paths $h = h_0 \rightarrow a_1 h_1 \ldots h_k$ and for all paths $\mathcal{E} (0) \rightarrow a_1 \mathcal{E} (1) \ldots \rightarrow a_k \mathcal{E} (k)$ such that $a_1, \ldots, a_k \in G$ and $\text{in}_{\mathcal{E} (i)} \in h_i$, we have $(\mathcal{E} (k) G \varphi) \in h_k$ by iterating point $(C_G)$ of Definition 26. By 25, we have $(\mathcal{E} (k) \varphi) \in h_k$. By $\mathcal{P}(\text{in}_\mathcal{E} (k))$, we have that $\mathcal{M} \otimes \mathcal{E} (k), (h_k, \mathcal{E} (k) \varphi) \models \varphi$. Thus $\mathcal{M} \otimes \mathcal{E} (k), h^e \models C_G \varphi$.

- Suppose that for all $\mathcal{E}$ of length $n$, we have $\mathcal{P}(\text{in}_\mathcal{E} \varphi), \mathcal{P}((\mathcal{E} \varphi))$. Let $\mathcal{E}$ be of length $n$ and let us prove that $\mathcal{P}((\mathcal{E} \langle E_0 \rangle \varphi))$.

  $(\Rightarrow)$ Suppose that $(\mathcal{E} \langle E_0 \rangle \varphi) \in h$. It implies that for all paths $h = h_0 \rightarrow a_1 h_1 \ldots h_k$ and for all paths $\mathcal{E} (0) \rightarrow a_1 \mathcal{E} (1) \ldots \rightarrow a_k \mathcal{E} (k)$ such that $a_1, \ldots, a_k \in G$ and $\text{in}_{\mathcal{E} (i)} \in h_i$, we have $(\mathcal{E} (k) \varphi) \in h_k$ by iterating point $(C_G)$ of Definition 26. By 25, we have $(\mathcal{E} (k) \varphi) \in h_k$. By $\mathcal{P}(\text{in}_\mathcal{E} (k))$, we have that $\mathcal{M} \otimes \mathcal{E} (k), (h_k, \mathcal{E} (k) \varphi) \models \varphi$. Thus $\mathcal{M} \otimes \mathcal{E} (k), h^e \models C_G \varphi$.

$(\Leftarrow)$ Suppose that $(\mathcal{E} \langle E_0 \rangle \varphi) \notin h$. By Definition 25, we have $(\mathcal{E} \langle E_0 \rangle \varphi) \notin h$, where $\neg C_G \varphi$ is the negation of $C_G \varphi$. Step $C_G$ in the Algorithm of Figure 5.2 ensures the existence of a path $h = h_0 \rightarrow a_1 h_1 \ldots h_k$ and a path $\mathcal{E} (0) \rightarrow a_1 \mathcal{E} (1) \ldots \rightarrow a_k \mathcal{E} (k)$ such that $a_1, \ldots, a_k \in G$ with $\text{in}_{\mathcal{E} (i)} \in h_i$ and $(\mathcal{E} (k) \varphi) \in h_k$. In other words, $(\mathcal{E} (k) \varphi) \notin h_k$. By $\mathcal{P}((\mathcal{E} (k) \varphi))$, $\mathcal{M} \otimes \mathcal{E} (k), (h_k, \mathcal{E} (k) \varphi) \models \varphi$. Properties $\mathcal{P}(\text{in}_\mathcal{E} \varphi)$ ensure the path is indeed in the product model $\mathcal{M} \otimes \mathcal{E} |_h$, thus $\mathcal{M} \otimes \mathcal{E} |_h, (h, \mathcal{E} \varphi) \models C_G \varphi$.

We conclude by applying the truth lemma (Lemma 2) to the Hintikka set $h$ that contains $(\epsilon \Phi)$ and we obtain that $\mathcal{M}, h \models \Phi$. ■
CHAPTER 5. SATISFIABILITY PROBLEM

Figure 5.3: (Expected) Kripke model that represents the computation tree of $M$ on the input instance $\omega$.

Remark 1 The $2\text{EXPTIME}$ upper bound also holds for the satisfiability problem of $\text{DELCK}$ in $S5$ Kripke models. We proceed as in [HM92] (p. 358). We add the following clauses to definition 25:

\[(5') \text{ If } (\vec{e} K_a \varphi) \in h \text{ then } (\vec{e} \varphi) \in h;\]

We add the following clause in the definition of $R_a$ in Definition 26:

for all $\vec{e} \rightarrow^a \vec{e}'$, if $\text{in}_{\vec{e}} \in h$ and $\text{in}_{\vec{e}'} \in h'$ then $(\vec{e} K_a \varphi) \in h$ iff $(\vec{e}' K_a \varphi) \in h'$.

5.4.2 Reduction

The aim of this section is to prove the following theorem.

Theorem 4 The satisfiability problem of $\text{DELCK}$ is $2\text{-Exptime}$-hard.

Let us consider any $2\text{-Exptime}$ decision problem $L$. As $\text{AEXPSPACE} = 2\text{-Exptime}$ [CS76], it is decided by an alternating Turing machine $M$ that runs in exponential space. W.l.o.g we suppose that all executions halt and no state is a negated state. We will define a polynomial reduction $tr$ from $L$ to the satisfiability problem of $\text{DELCK}$, that is $tr$ will be computable in polynomial time, and $\omega$ is a positive instance of $L$ if and only if $tr(\omega)$ is a satisfiable $\text{DELCK}$-formula.

The idea of $tr(\omega)$ is to enforce an expected form of a Kripke model as shown in Figure 5.3 that represents the computation tree of $M$ starting with $\omega$ on the tape. The cursor of the machine remains in the $N$-first cell portion of the tape, where $N$ is exponential in $|\omega|$. We define $N_0 = \log_2(N)$ for the rest of the section. $N_0$ is polynomial in $|\omega|$. We introduce two agents: agent $ex$ for the transitions in the computation tree and agent $t$ for the linear structure of tapes. A configuration of the Turing machine is represented by a sequence of worlds linked by agent $t$: one so-called control world followed by cell worlds.

\footnote{If not, we add a double exponential counter to the machine and we abort the execution after a double exponential number of steps.}
5.4. COMMON KNOWLEDGE

- The control world contains the type of the configuration: existential (resp. universal) if \( p_{\exists} \) (resp. \( p_{\forall} \)) is true. A special atomic proposition \( p_{\text{win}} \) tags control worlds that correspond to winning configurations for player \( \exists \).

- Cell worlds represent the cells of the tape and form a linear structure. They are indexed by \( x \) from \( x = 0 \) (left-most cell) to \( x = N \) (right-most cell). In each cell world, \( p_{a} \) is true means that the corresponding cell contains letter \( a \in \Gamma \). A proposition of the form \( p_{q} \) being true means that the cursor is at that cell and the current state is \( q \in Q \).

Besides atomic propositions \( p_{\exists}, p_{\forall}, p_{a}, a \in \Gamma \) and \( p_{q}, q \in Q \), we also consider the list of atomic propositions for the bits of the cell index \( x: x_{1}, \ldots, x_{N_{0}} \). We also consider another such list for another cell index \( v: v_{1}, \ldots, v_{N_{0}} \). The index \( v \) will be used to compare cell worlds of tapes of a configuration and a successor configuration different tapes during transitions.

The definition of \( tr(\omega) \) needs multi-pointed event models \((E_{i}^{t}, E_{0}^{t})\) given in Figure 5.4 non-deterministically and publicly choose the \( i^{th} \) bit of value \( v \). We also consider Boolean formulas \( x \leq v, x = v, x = v - 1 \) and finally \( K_{i}x = x + 1 \) (the value of \( K_{i}x_{0} \ldots K_{i}x_{N_{0}} \) is equal to \( x + 1 \)). All the formal definitions are given in \([CS18]\). We define the abbreviation \([\text{choose} v] = [E_{0}^{t}, E_{0}^{t}] \ldots [E_{N_{0}}^{t}, E_{N_{0}}^{t}]\). Technically, it corresponds to non-deterministically choosing and publicly announcing a value for \( v \).

**Definition 27** Formula \( tr(\omega) \) is the conjunction of the formulas shown in Table 5.7.

In formulas of Table 5.1, common knowledge operators \( C_{ex} \) and \( \hat{C}_{ex} \) are used to talk about any control world, while \( C_{t} \) and \( \hat{C}_{t} \) talk about any cell world.

Importantly, notice that with \( \text{DELCK} \), it is impossible to force the each world to have exactly one successor. Thus in general, the expected Kripke model is not as depicted in Figure 5.3. Instead, we ensure that cell worlds of depth \( k \) have the value \( x = k \). We use formula (9) that imposes the value of \( x \) to be the same in all successor cell worlds, and formula (10) saying that everywhere, \( K_{i}x = x + 1 \). Formula (11) states that only one \( p_{a} \) is true in each cell world, formula (12) states that only one \( p_{q} \) is true in some cell world, and formula (13) states that if \( p_{q} \) is true in a cell world, then no \( p_{q} \) is true in all the \( t \)-successors. Formulas (14), (15), (16) and (19) define the initial tape. Transitions are ensured by formulas (14) to (16). These formulas automatically ensure that several cell worlds with the same index \( x \) have the same valuation over \( x, p_{a}, a \in \Gamma, p_{q}, q \in Q \).

Formulas that handle transitions use integer \( v \) to pinpoint a cell index in the tape. It is used in formula (14) to tell that when the cursor is not in a cell world, then the letter remains the same during any transition. It is also used in formula (15) to check the existence of all compatible transitions and in formula (16) to check that all successor control worlds and their tapes correspond to a transition.

**Proposition 8** \( tr(\omega) \) is satisfiable if and only if \( \omega \) is a positive instance of \( L \).

**Proof.**

\((\Rightarrow)\) Suppose \( tr(\omega) \) is satisfiable. There is a pointed Kripke model \( M, w \) that satisfies \( tr(\omega) \). The model is not exactly as in Figure 5.3 for instance, a cell world can have several successors but that have all the same valuation. We extract the computation tree of \( M \) on the
Valuations for control worlds

1. \( C_{ex}(x = 0) \)
2. \( C_{ex}(p_3 \leftrightarrow \bigvee_{q \in \Gamma}(q \rightarrow \tilde{C}_t p_q))\) and \( C_{ex}(p_3) \)
3. \( C_{ex}(\bigwedge_{a \in \Gamma} \neg p_a \land \bigwedge_{q \in Q} \neg p_q) \)

Winning condition

If the current state is \( accept \) the world is marked as winning.

If the current state is \( reject \) the world is marked as losing.

If the current state is not \( accept \) and is universal, the world is marked as winning if all successor worlds are marked as winning.

If the current state is not \( accept \) and is existential, the world is marked as winning if one successor world is marked as winning.

Tape

The cell index of the left-most cell is 0.

On any tape world, the value of \( x \) is the same in all successors.

On any tape world, the value of \( x \) is incremented by 1 on all successors.

On any tape world, only one \( p_a \) is true and represent the current letter on the cell.

On any tape, somewhere only one \( p_a \) is true.

Anywhere, if \( p_a \) is true then no \( p_{a'} \) is true anywhere on the rest of the tape.

Transitions

We define here \( \varphi(q,a,q',b,d) = C_t((x = v \rightarrow p_0 \land \neg p_2) \land (x = v + d \rightarrow p_{a'})) \)

On the tape, if no \( p_a \) is true and \( p_a \) is true, then at the same position on the successors’ tapes, \( p_a \) is true.

If there is a transition it must be present on the model.

In every world, any \( ex \)-successor must correspond to a transition.

Initial configuration

The letters of the initial word are on the initial tape.

Cells of index \( \omega \) are blank.

Head in the left-most cell. Initially in the initial state.

The initial control world is winning.

Table 5.1: Clauses of \( DELCK \)-formula \( tr(\omega) \).
initial configuration on input $\omega$. We prove by induction that $p_{\text{win}}$ tags a control world iff its corresponding configuration is winning. Thus, as the initial configuration is tagged by $p_{\text{win}}$, the input $\omega$ is accepted by $M$.

($\Leftarrow$) Suppose that $\omega$ is a positive instance of $L$. We construct the model $M, w$ that represent the computation tree of $M$ on input $\omega$ as shown in Figure 5.3. By construction, formula $M, w \models tr(\omega)$. ■

The lower bound given in Theorem 4 still holds for the variant of the satisfiability problem where we require the model to be $S5$, that is, epistemic relations, to be equivalence relations (see [CS18] for details).

5.5 Arbitrary events

The satisfiability problem of arbitrary public announcement and variants is undecidable ([FvD08], [AvDF14], [AvDF16], [vDvdHK16]). Interestingly, the existential fragment of arbitrary public announcements is decidable [vDFH].

Surprisingly, the satisfiability problem of refinement modal logic (which corresponds to epistemic logic extended with an ‘arbitrary event model operator’) is decidable and proven to be non-elementary [BvDF$^{+}$14].

5.6 Implementation

The tableau method for EL is implemented in Hintikka’s world and is based on the tableau prover Mettel2 [TSK12].
Chapter 6

Epistemic planning

In artificial intelligence, classical planning consists in generating a finite sequence of actions for achieving a given goal. The so-called epistemic planning problem (introduced in [BA11], nicely explained in [Hol17]) generalizes classical planning with epistemic goal (agent $a$ knows that...) and complex actions (public announcements, private announcements, etc.). Epistemic planning based on Dynamic epistemic logic ([BMS98a], [Van07]) is undecidable in the general case [BA11] as we will see in this chapter. Formally, we consider the following decision problem:

**Definition 28 (Epistemic planning)**

- input: a state $S$, a finite set of actions $\mathbb{A}$ and a formula $\varphi \in \mathcal{L}_{\text{ELCK}}$;
- output: yes if there exists a sequence of actions (a plan) $A_1, \ldots, A_m \in \mathbb{A}$ such that $S \otimes A_1, \ldots, A_m \models \varphi$; no otherwise.

6.1 Undecidability proof

We have:

**Theorem 5** Epistemic planning is undecidable.

The idea for proving the undecidability of epistemic planning is to reduce the halting problem of Turing-complete machines. The reduction works according to the following correspondence table:

<table>
<thead>
<tr>
<th>For the halting problem</th>
<th>For the epistemic planning problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial configuration</td>
<td>Initial epistemic model</td>
</tr>
<tr>
<td>Configuration</td>
<td>Epistemic model</td>
</tr>
<tr>
<td>Machine transition specification</td>
<td>Repertoire of actions</td>
</tr>
<tr>
<td>Transition application</td>
<td>Update product</td>
</tr>
<tr>
<td>Halting configuration</td>
<td>Specified by the goal</td>
</tr>
</tbody>
</table>

In this section, we give the simplest proof we know of Theorem 5 inspired by [CPS18] based on cellular automata. Actually, we show the undecidability of an epistemic planning problem, where $\mathbb{A}$ is fixed and not part of the input. For this, we rely on a universal cellular automaton. From now on throughout this section, $AGT = \{a, b\}$; recall that all models (i.e., states and actions) are $S5$ (epistemic relations are equivalence relations).
CHAPTER 6. EPISTEMIC PLANNING

6.1.1 Universal 1D cellular automata

In this section, we only consider one-dimensional three-neighbor cellular automata. An infinite sequence of cells are settled on a line; each cell is in a state represented by a symbol of a finite alphabet \( \Sigma \). Given a cell, a transition function \( f \) maps a three-neighbor (left-cell symbol, current symbol, right-cell symbol) to the new symbol of the cell.

Definition 29 A cellular automaton is a pair \( A = (\Sigma, f) \) where \( \Sigma \) is a finite alphabet and \( f : \Sigma^3 \to \Sigma \) is a transition function.

Example 33 (Rule 110 [Wol02]) The Rule 110 cellular automaton is the two-symbol cellular automaton \( A_{R110} = (\{0, 1\}, f_{110}) \) where \( f_{110} \) is defined by the Boolean formula
\[
\begin{align*}
&\text{x} \land \text{y} \land \neg \text{z} \\
&\lor \text{x} \land \neg \text{y} \land \text{z} \\
&\lor \neg \text{x} \land \text{y} \land \text{z} \\
&\lor \neg \text{x} \land \text{y} \land \neg \text{z} \\
&\lor \neg \text{x} \land \neg \text{y} \land \text{z} \\
\end{align*}
\]

A configuration, that is the symbols of cells on an infinite line, is modeled by an infinite word \( c \in \Sigma^\mathbb{Z} \), that is, a map that assigns a symbol \( c[i] \) to any integer \( i \in \mathbb{Z} \). A computation step is performed by the following rule.

Definition 30 Given a cellular automaton \( A \), given an infinite word \( c \in \Sigma^\mathbb{Z} \), we define the successor of \( c \) by \( A \) to be the infinite word \( c' \) defined by \( c'[i] := f(c[i-1], c[i], c[i+1]) \). We write \( c \rightarrow_A c' \).

Figure 6.1 shows the transition function \( f_{110} \) graphically and some successive configurations.

A cellular automaton is deemed universal if it can simulate any Turing machine; the quest for finding such small universal cellular automata started in the 1960s. A common hypothesis is to assume a blank background: we consider that alphabets always contain a special symbol \( \_ \) and that transition functions map \( \_ \) to \( \_ \). Furthermore, we assume that configurations are finite, in the sense that almost all cell symbols are \( \_ \) except a finite number – configurations are of the form \( \_ \alpha \omega \) where \( \alpha \) is a finite word, called the support of the configurations. Starting from a finite configuration only leads to finite configurations.

Smith proved that any \( m \)-symbol \( n \)-state Turing machine can be simulated by a one-dimensional \( (m + 2n) \)-symbol four-neighbor cellular automaton with a blank background (see Theorem 4 in [Smi68]). As Minsky constructed a 4-symbol 7-state universal Turing machine \( M_{\text{Minsky}} \) [Min67], there exists a \( 4 + 2 \times 7 = 18 \)-symbol universal cellular automaton \( A_{\text{Smith}} = (\Sigma_{\text{Smith}}, f_{\text{Smith}}) \), that simulates \( M_{\text{Minsky}} \).

As a consequence, we obtain the following undecidability result for the reachability problem for \( A_{\text{Smith}} \) with blank background.

---

1 We use ‘symbol’ instead of ‘cell state’, to avoid confusion with a knowledge state.

2 Referred to as \( (m + 2n) \)-state in [Smi68].
6.1. UNDECIDABILITY PROOF

**Theorem 6** There exists a finite word $h_{\text{Smith}}$ such that it is undecidable to determine, given a finite word $\alpha$, whether $\omega^{\omega} \alpha \omega^{\omega} \rightarrow_{A_{\text{Smith}}} c$ where the configuration $c$ contains the pattern $h_{\text{Smith}}$.

### 6.1.2 Finite linear states

The main result of this section is Corollary which provides bounds on the parameters of the epistemic planning problem. This is achieved by simulating executions of cellular automata with blank background. First of all, we introduce sufficiently many propositions to encode symbols of the alphabet. Typically, for alphabet $\Sigma$ with $18$ symbols that we respectively denote by $\ell_0, \ell_1, \ldots, \ell_{17}$, only $5$ propositions $p_1, \ldots, p_5$ suffice. Given a symbol $\ell$, we denote by $\text{enc}(\ell)$ the encoding of $\ell$: $\text{enc}(\ell_0) = \neg p_1 \land \ldots \neg p_5$, $\text{enc}(\ell_1) = p_1 \land \neg p_2 \ldots \neg p_5$, etc. W.l.o.g., we suppose that symbol $\omega$ is $\ell_0$ and is encoded by the valuation making all $p_i$’s false. In the rest, $\bar{p}$ denotes the sequence of propositions $p_i$’s.

Now, we encode the finite supports of configurations by means of finite linear states — such states appear in real epistemic puzzles such as the consecutive number puzzle $\text{VDK15}$. They are states of the form $I_a = \{(-n, \ldots, -1, 0, 1, \ldots, n)\}_{a \in A GT, w, 0}$ with odd $n$ and:

1. $R_a = \{(k, k) \mid k \in [1; n]\} \cup \{(2k, 2k + 1), (2k + 1, 2k) \mid -\frac{n+1}{2} \leq k \leq \frac{n-1}{2}\}$;
2. $R_b = \{(k, k) \mid k \in [1; n]\} \cup \{(2k, 2k - 1), (2k - 1, 2k) \mid -\frac{n+1}{2} \leq k \leq \frac{n-1}{2}\}$;
3. $\preceq \in w(k)$ iff $k$ is even;
4. for all $i$, $p_i \not\in w(-n), w(-n + 1), w(n - 1), w(n)$.

A finite linear state $I_n$ encodes a finite configuration $c$ whose support is of length smaller than $2n - 3$ when $V(k)$ makes $\text{enc}(\ell_i)$ true iff $c[k] = \ell_i$; we require that the two terminal ($-n$ and $n$) and the two pre-terminal ($n - 1, -n + 1$) worlds encode $\omega$ (Condition 4). For instance, the $A_{\text{Smith}}$ configuration $\omega^\omega \ell_0 \ell_5 \ell_2 \omega^\omega$ where $c[-1] = 9$ can be represented by:

$$
\begin{array}{cccccccc}
\preceq & b & -2 & a & -1 & b & 0 & a & 1 & b & 2 & a & 3 \\
\preceq & p_3 & p_4 & \preceq & p_1 & p_3 & \preceq & p_2 & \preceq
\end{array}
$$

Given a finite word $\alpha$, let us set once and for all what finite linear state $S_\alpha$ will encode $\alpha$: substituting $\alpha^m$ for $\alpha$ with $m \in \{0, 1, 2, 3\}$ so that $|\alpha|$ be congruent to $3$ modulo $4$, we set $S_\alpha = (I_{|\alpha| - 2}, (R_a)_{a \in AGT, w, 0})$ where $V(k - \frac{|\alpha| - 1}{2})$ makes $\text{enc}(\ell_i)$ true iff $\alpha[k] = \ell_i$ for $0 \leq k \leq |\alpha| - 1$, and makes $\text{enc}(\omega)$ true outside of this range. These are only technicalities ensuring a sufficiently large odd index for the interval state to respect constraints with a pseudo-centered word.

### 6.1.3 Simulating cellular automata in DEL

We define action $F$ mimicking one computation step of the cellular automaton: if $S$ is a finite linear state encoding a configuration $c$, then $S \otimes F$ is (isomorphic to) a finite linear state encoding the successor of $c$.

Action $F$ is partially given by Figure 6.2. Intuitively, the actual event $e_0$ copies every non-terminal world; event $e_{-1}$ keeps the left-tip world, while $e_{-2}$ and $e_{-3}$ clone it to append two new worlds to the left. Events $e_1$, $e_2$, $e_3$ play a similar part. In the end, action $F$ adds two new worlds on each side, while preserving the canonical knowledge state structure that we aim for, including the tips’ asymmetry relatively to the agents.

---

3According to the notation of Table 14.8-1 p. 279 in [Min67], word $h_{\text{Smith}}$ is $q_0^0$. 

We finish the definition of \( F \) by adding postconditions for \( p_j \)'s, corresponding to the application of a transition function \( f \). Suppose w.l.o.g. that \( \owns \) holds in a given world \( k \in \{-n+1, \ldots, n-1\} \). Bits of \( c[k-1] \) are obtained by taking the \( b \)-transition from the \( \neg \owns \)-world of world \( k \). They are: \( \langle \hat{K}_a(\neg \owns \land p_i) \rangle_i \). In the same way, the values of \( \vec{p} \) in world \( k+1 \) are given by the vector \( \langle \hat{K}_a(\neg \owns \land p_i) \rangle_i \). The case where \( \owns \) does not hold in the current world is symmetric.

We model \( f \) by Boolean formulas \( f_j(\vec{p}^-, \vec{p}, \vec{p}^+) \) over three sequences of atomic propositions \( \vec{p}^- \) (left cell symbol), \( \vec{p} \) (middle cell symbol), \( \vec{p}^+ \) (right cell symbol) that return the value of the \( j \)-th bit of the new symbol at the middle cell. Bits of the new symbol are: \( \langle f_j(\vec{p}^-, \vec{p}, \vec{p}^+) \rangle_j \).

The postconditions for \( p_j \)'s in \( F \) are thus defined as follows. First, \( \text{post}(e_0)(p_j) = \top \) for all \( k \neq 0 \). Only \( e_0 \) effectively applies \( f \) and \( \text{post}(e_0)(p_j) \) is the formula

\[
\left( \owns \rightarrow f_j(\langle \hat{K}_a(\neg \owns \land p_i) \rangle_i, \vec{p}, \langle \hat{K}_a(\neg \owns \land p_i) \rangle_i) \right) \land \left( \neg \owns \rightarrow f_j(\langle \hat{K}_a(\owns \land p_i) \rangle_i, \vec{p}, \langle \hat{K}_a(\owns \land p_i) \rangle_i) \right).
\]

We will refer to \( F_{\text{Smith}} \) for the action model that corresponds to the transition function \( f_{\text{Smith}} \) of the cellular automaton \( A_{\text{Smith}} \).

On top of proposition \( \owns \), when considering the particular cellular automaton \( A_{\text{Smith}} \), we need no more than 5 extra propositions to encode all symbols of alphabet \( \Sigma_{\text{Smith}} \) of cardinal 18, so that an overall set of 6 propositions suffices.

Now, we define formulas encoding the occurrence of a pattern \( h \) in the configuration. We first define the formula \( \text{wenc}(h) := \text{wenc}_{\owns}(h) \lor \text{wenc}_{\owns}(h) \) where formulas \( \text{wenc}_{\owns}(h) \) and \( \text{wenc}_{\owns}(h) \) are inductively defined by:

- \( \text{wenc}_{\owns}(e) = \owns \land \text{wenc}_{\owns}(e) = \neg \owns \);
- for all letters \( \ell \), \( \text{wenc}_{\owns}(lh) = \owns \land \text{enc}(\ell) \land \hat{K}_a \text{wenc}_{\owns}(h) \);
- for all letters \( \ell \), \( \text{wenc}_{\owns}(lh) = \neg \owns \land \text{enc}(\ell) \land \hat{K}_a \text{wenc}_{\owns}(h) \).

We can now state the following theorem that derives from our ability to simulate the behavior of the cellular automaton \( A_{\text{Smith}} \) and the use of the dual of the common knowledge operator \( \hat{C}_{\text{AGT}} \) to search over a finite linear state for the pattern \( \text{wenc}(h_{\text{Smith}}) \).

**Theorem 7** Given a finite linear state \( S \) over 6 propositions, it is undecidable to determine whether or not a state satisfying \( \hat{C}_{\text{AGT}} \text{wenc}(h_{\text{Smith}}) \) is reachable from \( S \) by executing a finite sequence of actions \( F_{\text{Smith}} \).

**Proof.**

By reduction from the undecidable reachability problem of Theorem 6, An instance \( a \) of the latter is translated into \( S_{\alpha} \).

**Corollary 1** The epistemic planning problem over 2-agent SS finite models is undecidable, even if the repertoire is \( \{ F_{\text{Smith}} \} \), the goal is \( \hat{C}_{\text{AGT}} \text{wenc}(h_{\text{Smith}}) \) and we only use at most 6 different propositions.

The interested reader can run simulations in DEL of cellular automata using the online software Hintikka’s world. Figure 6.3 shows a screenshot of Hintikka’s World depicting a finite linear state.

---

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th></th>
<th></th>
<th>b</th>
<th>a</th>
<th>c</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-( \hat{K}_a ) ▽</td>
<td>pre-( \hat{K}_a ) ▽</td>
<td>pre-( \hat{K}_a ) ▽</td>
<td>pre-( \hat{K}_a ) ▽</td>
<td>pre-( \hat{K}_a ) ▽</td>
<td>pre-( \hat{K}_a ) ▽</td>
<td>pre-( \hat{K}_a ) ▽</td>
<td>pre-( \hat{K}_a ) ▽</td>
<td></td>
</tr>
<tr>
<td>post-( \owns = \top )</td>
<td>post-( \owns = \top )</td>
<td>post-( \owns = \top )</td>
<td>post-( \owns = \top )</td>
<td>post-( \owns = \top )</td>
<td>post-( \owns = \top )</td>
<td>post-( \owns = \top )</td>
<td>post-( \owns = \top )</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.2: Skeleton of action \( F \).
6.2 Undecidability frontier

Epistemic planning is undecidable even when we assume these conditions on event models:

- preconditions and postconditions (even up to modal depth 1) and no common knowledge in the goal formula [BA11, CPS18];
- preconditions up to modal depth 2, no postconditions and no common knowledge in the goal formula [CMS16];
- arbitrary preconditions, no postconditions [AB13] and no common knowledge in the goal formula;
- automatic (potentially infinite) initial state but only one public actions [CPS18].

Conjecture 1 Epistemic planning when actions are purely epistemic and all preconditions are of modal depth at most 1 is decidable.

6.3 Decidable restrictions

6.3.1 Public actions

Theorem 8 Epistemic planning with public actions is in PSPACE.

Proof.
We give a non-deterministic algorithm for deciding epistemic planning with public actions in polynomial space. We start with the initial Kripke model as the current model. At each step, we check whether the goal is satisfied. If yes, we accept the input. If not, we non-deterministically
choose an action and we compute the product of the current model and the selected action. As actions are public, the current model remains of polynomial size in the input. ■

6.3.2 Purely epistemic actions

As public announcement of propositional facts commute and are idempotent, we have:

**Theorem 9** Epistemic planning with propositional public announcements is NP-complete [BJS15b].

As purely epistemic propositional event models commute and an event model $E$ stabilizes at iteration $|E|^d$ with respect to bisimulation up to modal depth $d$ (see [CMS16]). Therefore, we guess a plan by:

- guessing how many times (at most $|E|^d$) each event is executed, where $d$ is the modal depth of the goal formula;
- using the model checking algorithm over the language with iterated event models where numbers are written in binary and that runs in polynomial space.

As this algorithm runs in polynomial space, we get:

**Theorem 10** Epistemic planning with purely epistemic propositional event models is PSPACE-complete ([BJS15b], [CMS16]).

6.4 Related work

Recently, connections have been made between epistemic planning and reachability games in graphs with imperfect information [MPS19]. In this setting, each player has a finite collection of event models in his repertoire. The existence of uniform strategies for some players against the others (considered as opponents) is studied. In future work, this setting has to be compared with implicit coordination in epistemic planning ([EBMN17]).

Interestingly, there exists a more syntactic approach to epistemic planning, referred as MEP (Multi-agent epistemic planning) (see [MBF+15], [HFWL17]). MEP works as follows:

- the initial state is described by epistemic formulas;
- actions are described in a STRIPS-like fashion, but with epistemic formulas instead of Boolean formulas;
- the goal is an epistemic formula.

The main difference with our epistemic planning (Definition 28) is that higher-order knowledge is described in the real world, contrary to DEL, in which preconditions/postconditions in events may be evaluated in any worlds. Thus, the state-space is finite and the computation of a plan highly relies on the satisfiability problem. For some syntactic restrictions, we can even reduce to classical planning [MBF+15]. In other words, MEP is decidable, even with common knowledge in the language [LL18].
Chapter 7

Propositional Epistemic planning

This chapter is adapted from DTPS18

7.1 Introduction

When event preconditions and postconditions are propositional (Boolean) only (intuitively, announcements are Boolean formulas and uncertainty is only about the content of the messages and physical changes), epistemic planning has been shown to be decidable ([YWL13], [Mau14]).

Regrettably, epistemic planning as considered so far concerns finite plans, and is in essence a reachability problem. However, real-life applications may challenge infinite plans. We give three examples of planning goals that require an infinite plan.

1. safety properties, e.g. ‘invariantly, an intruder \(a\) does not know the location of the piece of jewelry during more than 3 consecutive steps’.

2. recurring bounded properties, e.g. ‘all drones know that the region is safe every 20 steps’;

3. strategic reasoning, e.g. ‘with the current plan, the drone \(a\) never knows that the region is safe but every 10 steps, there is a(nother) plan to let the drone \(a\) eventually know that the region is safe’.

Nevertheless, infinite plans have been considered in [Mau14] in the DEL (Dynamic Epistemic Logic) setting. In this approach, DEL structures, those arising from iterating ad infinitum the triggering of events, are seen as infinite relational structures. Maubert has shown that, when all event preconditions and postconditions are propositional, DEL structures are automatic structures (over strings) [BG00, Rub08]. Automatic structures are relational structures which have a presentation by means of a finite family of finite-state automata. As noticed by Maubert, the decidability result of epistemic planning (for epistemic goals) relies on the fact that model checking against first-order logic is decidable over automatic structures: in a DEL structure, the skeleton is a tree whose nodes (or equivalently finite branches) are finite plans, i.e. sequences of triggered events, and this tree structure is augmented with “transverse” binary relations between nodes in possibly different branches. These extra relations denote epistemic relations between histories. In DEL structures, the existence of a plan reduces to the existence of a point in this automatic (tree) structure where the epistemic planning goal (hence expressible in FO) holds. After all, the whole epistemic planning problem can be expressed in FO whose outermost
quantifier is existential. Additionally, automata constructions that answer the query of an FO-formula on an automatic structure allow to synthesize an automaton that recognizes the set of all (finite) plans achieving the goal. Regarding infinite plans, Maubert relies on the involved theory of uniform strategies he has developed [Mau14]. This theory enables him to deal with epistemic planning instances whose goals are expressed in the branching-time epistemic logic $\text{CTL}^\text{K}$, the extension of temporal logic $\text{CTL}^*$ [HT87] with epistemic modalities. The setting does lead to synthesizing infinite plans, but the synthesis relies on a bottom-up technique over the goal formula that prevents the setting from being extended to logics featuring fix-points; while statements (ii) and (iii) above cannot be expressed in $\text{CTL}^\text{K}$, a well-tuned logic with fix-point or monadic with second-order quantifiers would suffice.

In an attempt to enlarge the specification language for epistemic planning goals, that goes beyond $\text{CTL}^\text{K}$ while remaining decidable, one should observe that using the full monadic second-order logic (MSO) is hopeless: by [Tho90], model checking against MSO over the binary tree equipped with the transverse binary relation “equal level” is undecidable. Relation “equal level” is an instance of an agent epistemic relation where the agent does not observe anything from the current finite sequence of triggered events but its length. However, there are variants of MSO, where second-order quantifications are constrained enough to yield a decidable model checking problem over infinite trees of the kind of DEL structures. For example, relying on [HT87], and as depicted in Figure 7.1 there exist mainly three interpretations of the second-order quantifiers, yielding full MSO, path-MSO, and chain-MSO. The full MSO logic can be interpreted over arbitrary relational structures and the second-order quantifiers range over all subsets of the structure domain. On the contrary, the path-MSO logic is interpreted on (possibly infinite) trees and the quantifiers range over paths/branches of the tree. Finally, chain-MSO, also interpreted on trees only, requires the second-order quantifier to range over chains, that are subsets of paths [HT87]. It is known that the expressive power of chain-MSO strictly subsumes the one of path-MSO, the former allowing to describe arbitrary $\omega$-regular properties of branches, while the latter captures only star-free properties, see [HT87].

In this chapter, we extend chain-MSO of [Tho87, HT87] with relations in the tree, e.g. interpreting epistemic modalities, making statements (ii) and (iii) expressible. This extension is written $\text{cMSO}$. We introduce regular automatic trees, a strict but large subclass of automatic structures that encompass DEL structures with propositional events. We then show that model checking against $\text{cMSO}$ is decidable over regular automatic trees. Our decidability result is strongly inspired from the proof technique in [Tho92, Th. 5.2] to show that chain-MSO with the “equal level” predicate is decidable over $n$-ary trees. Our proof yields automata constructions that can be exploited to achieve the synthesis of plans in epistemic planning.

Noticeably, the logic $\text{cMSO}$ captures the linear-time mu-calculus (BKP86, Var88a) enriched with path quantifiers and epistemic modalities, here written $\text{BL}^{\text{lin}\text{µ}}\text{K}$(the acronym stands for ‘Branching Logic of the linear-time $\mu$-calculus with Knowledge). In essence, the logic $\text{BL}^{\text{lin}\text{µ}}\text{K}$ is an epistemic extension of the logic $\text{ECTL}^*$, for “extended CTL$^*$”, of [HT87], but in a mu-calculus style rather than with the use of automata modalities, in the sense of [Wol83]. Since $\text{CTL}^\text{K}$ can easily be embedded into $\text{BL}^{\text{lin}\text{µ}}\text{K}$ (the hard-coded linear-time temporal operators of $\text{CTL}^*$ are based on fix-points expressible in the linear-time mu-calculus), our result significantly exceeds the one of [Mau14] whose proof technique does not allow for considering arbitrary fix-point formulas.

The presented contribution is derived from the preliminary work available in [Dou15]. In Section 7.2 we recall the notion of automatic structures and results on model checking against FO-formulas and MSO-formulas. In Section 7.3 we present the logic $\text{cMSO}$, as well as the subclass of so-called regular automatic trees. We prove that model checking against $\text{cMSO}$ is
7.2 Preliminaries on automatic structures and decidable theories

In this section we recall seminal results on automatic structures: namely that first-order logic \( \mathit{FO} \) is decidable over these structures while monadic second-order logic \( \mathit{MSO} \) is not. The interested reader may refer to [BG00, Rub08] for further details.

7.2.1 Structures and logics

Logics \( \mathit{FO} \) and \( \mathit{MSO} \) are interpreted over relation structures.

**Definition 31** A relational structure is a structure of the form \( A = \langle D, R_1 \ldots R_p \rangle \) where:

- \( D \) is a non-empty set called the domain;
- \( R_1 \ldots R_p \) are relations over \( D \) of arity \( r_1 \ldots r_p \), respectively; namely \( R_i \subseteq D^{r_i} \).

The set of symbols \( \{R_1 \ldots R_p\} \) is called the signature of \( A \). We take the convention to write \( R_i(d_1, \ldots, d_{r_i}) \) for \( (d_1, \ldots, d_{r_i}) \in R_i \). We distinguish the particular class of structures that are infinite trees of fixed finite degree \( n \), called \( n \)-ary trees, that are central in our contribution. These structures represent computation trees (with a root of address \( \epsilon \), \( i \)-successor relations, and prefix binary relation between addresses) augmented with additional relations, such as epistemic relations.

**Definition 32** Given a finite set \( E := \{1, \ldots, n\} \) of directions, we call \( n \)-ary tree a structure \( T = \langle D, r, S_1, \ldots, S_n, R_1, \ldots, R_p \rangle \) where:

- \( D \) is a prefix-closed subset of \( E^* \) (the addresses of the nodes);
- \( r \) is a unary relation that holds only for \( \epsilon \), the root address of the tree;
- \( S_e(x, xe) \) whenever \( xe \in D \);
R_1, \ldots, R_p are additional relations on D.

In the particular case of a tree, elements of D are nodes. Notice that no condition was imposed on the relations R_1 \ldots R_p in Definition 32 of a tree. They may therefore correspond to transversal links between nodes just as epistemic relations would do, e.g. the “equal level” binary relation.

Example 34 The structure T_2 = (E^*, S_1, S_2) is a 2-ary tree called the infinite full binary tree. Also the structure T'_2 = (E^*, S_1, S_2, \varepsilon) obtained by augmenting T_2 with the binary relation “at equal level in the tree” (\varepsilon(x, y)) holds if, and only if, |x| = |y| is another example of a 2-ary tree.

First-order logic (FO) over relational structures \mathcal{A} = \langle D, R_1 \ldots R_p \rangle concerns formulas that conform to the following syntax: \varphi ::= R_i(x_1 \ldots x_{r_i}) | \neg \varphi | (\varphi \land \varphi) | \exists x \varphi, where x, x_1, x_{r_i} are first-order variables whose interpretation ranges over the domain of relational structures.

When turning to the more expressive monadic second-order logic (MSO), one allows for second-order variables that range over subsets of the domain of relational structures. Formally, the syntax of MSO is given by: \varphi ::= R_i(x_1 \ldots x_{r_i}) | \exists x \varphi | (\varphi \land \varphi) | \exists x \varphi | \exists X \varphi, where x is a first-order variable, while X, X_1, \ldots, X_m are second-order variables whose interpretation ranges over subsets of the domain. In the following, we let V_1 and V_2 denote respectively the set of first-order variables and second-order variables.

As usual, in a formula, a variable x \in V_1 (resp. X \in V_2) is free if it is not under the scope of a quantifier, namely some Qx (resp. QX) with Q \in \{\forall, \exists\}. A formula is closed if it contains no free variables.

An assignment in domain D is a function \sigma that maps a variable of \mathcal{V}_1 onto an element of D and a variable of \mathcal{V}_2 onto a subset of D. Given an assignment \sigma in D, a variable x \in V_1 and d \in D, we let \sigma[x \mapsto d] be the assignment on D that coincides with \sigma but maps x onto d. Similarly, we define \sigma[X \mapsto D'] where D' \subseteq D for the second-order variables.

Due to limited space, we do not provide the semantics of MSO, see for example [EE01].

7.2.2 Automatic presentations

We now describe how some relational structures can be encoded using formal languages, following the presentation of [Rub08]. By alphabet we mean a finite set of symbols, named letters. A word over \Sigma is a finite sequence of letters. We denote by \Sigma^* the set of such sequences. For \alpha \in \Sigma^* and n \geq 0, let \alpha[n] be the n + 1-th letter of \alpha (when defined). We assume familiarity with the basic definitions of automata theory and the properties of regular languages.

Let \Box be a fresh padding symbol. If \Sigma is an alphabet, we define \Sigma_\Box as \Sigma \cup \{\Box\}. The following definition describes an encoding for k-tuples of words as a word on the product alphabet \Sigma_\Box^k; since words of the tuple may have different length, the padding symbol \Box is used to align the words.

Definition 33 If u, v \in \Sigma^*, their convolution u \otimes v is the word of (\Sigma_\Box \times \Sigma_\Box)^* of length \max(|u|, |v|) such that (u \otimes v)[n] = (u[n], v[n]) if n < \min(|u|, |v|); (u \otimes v)[n] = (u[n], \Box) if |v| \leq n < |u|; and (u \otimes v)[n] = (\Box, v[n]) if |u| \leq n < |v|. Convolution is defined similarly for k-tuples of words.

Example 35 The convolution of aaba, b and ba over alphabet \Sigma = \{a, b\} is the four-letter word \[ b \Box (\Box a) \Box (\Box a) \] over alphabet \Sigma_\Box^3.

Definition 34 Let \mathcal{A} = \langle D, R_1 \ldots R_p \rangle be a relational structure. An automatic presentation of \mathcal{A} is a tuple of regular languages (L_D, L_1, \ldots, L_n) meeting the following conditions:
7.2. PRELIMINARIES ON AUTOMATIC STRUCTURES AND DECIDABLE THEORIES

- that there exists a one-to-one encoding function \( \text{enc}: D \to L_D \);
- for all \( R_i \) (arity \( r_i \)), \( L_i = \{ \alpha_1 \otimes \cdots \otimes \alpha_{r_i} \mid \forall j, \alpha_j \in L_D \text{ and } (\text{enc}^{-1}(\alpha_1), \ldots, \text{enc}^{-1}(\alpha_{r_i})) \in R_i \} \).

The alphabet \( \Sigma \) of \( L_D \) is called the encoding alphabet. The inverse \( \text{enc}^{-1} \) of the encoding function is the decoding function. We write \( \text{enc}(D) \) for \( L_D \), and more generally, \( \text{enc}(R) \) for the set \( \{(\text{enc}(d_1), \ldots, \text{enc}(d_r)) \mid (d_1, \ldots, d_r) \in R \} \), for an arbitrary relation \( R \subseteq D^r \).

A structure is automatic if it has (at least) an automatic presentation. A presentation can effectively be described by a tuple of finite automata \((M_D, M_1, \ldots, M_n)\) recognizing \((L_D, L_1, \ldots, L_n)\).

Remark 2 We may assume that equality is among the relations \( R_i \), represented by the regular language \( \{ \alpha \otimes \alpha \mid \alpha \in L_D \} \). In the literature, the standard definition of automatic presentations allows an element to have several encodings, whenever equality can be presented by some regular language. However, both definitions enable to present the same structures [BG00].

Example 36 1. Finite structures are automatic, since finite languages are regular.

2. \((\mathbb{N}, \leq)\) is automatic, with an automatic presentation over a unary alphabet \( \{ 1 \} \) by letting \( \text{enc}(n) \) to be the unary representation of \( n \), that is the word \( 1^n \). Automaton \( M_\leq \) depicted in Figure 7.2 verifies there are less 1’s in the first component than in the second component.

![Figure 7.2: The finite-state automaton \( M_\leq \) of Example 36 (ii).](image)

3. Both trees \( T_2 \) and \( T_2^d \) are automatic.

We recall the known fundamental theorem of [Rub08, Th. 3.1] regarding the model checking problem against FO over automatic structures.

Definition 35 Model checking against FO

Input: A presentation \((M_D, M_1, \ldots, M_n)\) of a relational structure \( A \) and a closed FO-formula \( \varphi \) over the corresponding signature.

Output: Yes if \( A \models \varphi \), no otherwise.

Theorem 11 [Rub08, Th. 3.1] Model checking against FO is decidable.

The main ingredient of the proof of Theorem 11 relies on the construction of Proposition 9 which heavily relies on the closure of regular languages by intersection, negation and projection over components.

Let \( \varphi(x_1 \ldots x_n) \in \text{FO} \) where \( x_1 \ldots x_n \in V_1 \) are the only free variables, and let \( A \) be a relational structure. We write \( \varphi^A := \{(d_1 \ldots d_n) \in D^n \mid A, [x_i \mapsto d_i]_{1 \leq i \leq n} \models \varphi(x_1 \ldots x_n)\} \) for the set of tuples that satisfy \( \varphi \).

Proposition 9 There is an algorithm that given an automatic presentation of a relation structure \( A \) with encoding function \( \text{enc} \), and a first-order formula \( \varphi(x_1, \ldots, x_n) \), outputs an automaton that recognizes \( \text{enc}(\varphi^A) \).
Theorem 11 can sometimes be extended to MSO. For example, there are very specific cases where MSO can be decided, among which the typical example of the automatic structure $(\mathbb{N},<)$.

Proposition 10 [Bar07] The MSO-theory of a structure having an automatic presentation based on a unary encoding alphabet is decidable.

Also, by Rabin’s Theorem [Rab69], MSO is decidable over the full binary tree with signature $r,S_1,S_2$. However, MSO becomes undecidable over the full binary relation augmented with the “equal level” relation $el$ (see Example 34), although the obtained structure is automatic.

Theorem 12 [Tho90] Model checking against MSO over the binary tree $T_{2^n}$ is undecidable.

We now focus on a subclass of automatic structures where the undecidability frontier can be pushed back.

7.3 Model checking against cMSO over regular automatic trees

We consider the variant of MSO stemming from “chain MSO”, written cMSO in this paper, which contrary to FO and MSO, can only be interpreted over infinite trees. The second-order quantifiers range over chains, namely subsets of nodes along a branch, as depicted in Figure 7.1(c). The logic cMSO has already been defined in [Tho87, HT87, Tho92] for $n$-ary trees and with a signature restricted to the successor relations $S_1,\ldots,S_n$; the main results concern the decidability of the model checking of chain-MSO over the full binary tree, and its expressivity. In our setting, the signature of cMSO can be arbitrary, so that properties involving e.g. knowledge and time can be expressed.

We also exhibit a subclass of automatic structures called regular automatic trees that are automatic trees where the encoding of a node is its address in the tree. This subclass is large and encompasses infinite models arising from the unfolding of finite-state systems but also in dynamic epistemic logic. We establish that model checking against cMSO is decidable over regular automatic trees (Theorem 13).

7.3.1 The logic cMSO over trees

We recall the notion of tree structure $T = \langle D, r, S_1,\ldots,S_n,R_1,\ldots,R_p \rangle$ over a finite set $E = \{1,\ldots,n\}$ of directions (Definition 32), where $D$ is a prefix-closed subset of $E^*$ describing the nodes through their addresses in the tree. In the following, we let $S^*$ be the reflexive and transitive closure of the generalized successor relation $S := \bigcup_{i=1}^n S_i$.

Definition 36 In a tree structure $T$ of domain $D \subseteq E^*$, a subset $C \subseteq D$ is a chain if it is totally ordered with respect to $S^*$, i.e. for all $d,d' \in C$, either $S^*(d,d')$ or $S^*(d',d)$ holds.

We denote by $Ch(T)$ the set of chains of the tree $T$.

Example 37 In the full binary tree $T_2$, the set $\{1^{2^n} \mid n \in \mathbb{N}\}$ is a chain, whereas $\{1,2\}$ is not.

The notion of chains enables one to consider an interpretation of MSO formulas in trees, named cMSO, where the second-order quantifiers are restricted to chains.

Definition 37 The logic cMSO is interpreted in a tree $T = \langle D, r, S_1,\ldots,S_n,R_1,\ldots,R_p \rangle$ with an assignment $\sigma$ in $D$ for free variables as follows.
7.3. MODEL CHECKING AGAINST CMSO OVER REGULAR AUTOMATIC TREES

For a close formula \( \varphi \), we simply write \( \mathcal{T} \models \varphi \) without specifying the assignment.

The logic \textit{cMSO} is close to the logic \textit{path-MSO} \cite{Tho87,HT87} which restricts second-order quantification to maximal branches only. Actually the latter is a subsystem of the former: the \textit{cMSO}-formula

\[
\text{pathfrom}[X,x_0] := x_0 \in X \land \forall x \{ x \in X \implies [(\exists y S(x,y) \rightarrow \exists y (S(x,y) \land y \in X)) \land \neg S(x,x_0)] \}
\]

expresses that chain \( X \) is a maximal path starting at note \( x_0 \), which corresponds to the very quantifier in \textit{path-MSO}. This translation has already been noticed many times in the literature.

### 7.3.2 Regular automatic trees

We now turn to a particular class of trees, called \textit{regular automatic trees}, which are automatic trees for the most intuitive encoding of nodes, that is by their address in the tree. Since the domain is required to be regular, such trees arise from the unfolding of some finite-state transition systems, hence the terminology of \textit{regular trees}.

**Definition 38** A tree \( \mathcal{T} = \langle D, r, S_1, \ldots, S_n, R_1, \ldots, R_p \rangle \) on the set of directions \( E = \{1, \ldots, n\} \) is a \textit{regular automatic tree} if the identity function \( \text{enc} : D \rightarrow E^* \) describes an automatic presentation of \( \mathcal{T} \) over the encoding alphabet \( E \). This presentation is called the \textit{canonical presentation} of \( \mathcal{T} \).

Since the encoding function of a regular automatic tree is the identity function, the regularity of \( S_1, \ldots, S_n \) and \( \preceq \) directly follows from that of \( D \). Thus, a tree is a regular automatic tree if, and only if, \( D \subseteq E^* \) is regular, and for all \( R_i \) (of arity \( r_i \)), \( L_i = \{ d_1 \otimes \cdots \otimes d_{r_i} \mid (d_1, \ldots, d_{r_i}) \in R_i \} \)

is regular.

**Example 38** The tree \( T_{31}^e \) of Example 34 is a regular automatic tree.

As stated in the following proposition, regular automatic trees form a strict subclass of automatic trees.

**Proposition 11** There are automatic trees that are not regular automatic trees.

**Proof.**

Let \( E = \{1,2\} \) we consider the binary tree \( T_{\text{notreg}}^{\text{auto}} = \langle D, r, S_1, S_2, el \rangle \) of Figure 7.3(a) where \( D = \{1^m2^k \mid 0 \leq k \leq m \} \) and \( el \) is the “equal level” relation of Example 34. As \( D \) is not a regular language, \( T_{\text{notreg}}^{\text{auto}} \) is not a regular automatic tree; it is automatic: consider the encoding alphabet \( \Sigma = \{(0,1)\}_0^2 \) and let \( \text{enc}(1^m2^k) = \text{bin}(m) \otimes \text{bin}(k) \) be the encoding function, as presented in Figure 7.3(b) where \( \text{bin}(m) \) (resp. \( \text{bin}(k) \)) the binary representation of \( m \) (resp. \( k \)) with least significant digit first. For example, \( \text{enc}(112) = \text{enc}(1^22^1) = \text{bin}(2) \otimes \text{bin}(1) = \binom{1}{0} \binom{1}{0} \). The reader should easily get convinced that \( \text{enc}(D), \text{enc}(S_1), \text{enc}(S_2) \) and \( \text{enc}(el) \) are regular languages.

\[\square\]
CHAPTER 7. PROPOSITIONAL EPISTEMIC PLANNING

When restricting to regular automatic trees, some usual relations over trees can be easily captured by automata over the canonical encoding of nodes. We recall the following additional binary relations over trees.

- The reflexive and transitive closure $S^*$ of the generalized successor binary relation $S$;
- The binary relation $(d, d') \in \preceq$ whenever node $d$ is not deeper than node $d'$ in $T$;
- The binary relation $\mathsf{el}$, where $(d, d') \in \mathsf{el}$ whenever nodes $d$ and $d'$ are at the same depth in the tree.
- The binary equality relation $\mathsf{=}$.  

**Lemma 3** Let $\langle D, r, S_1, \ldots, S_n, R_1, \ldots, R_p \rangle$ be a regular automatic tree. Then the relational structure $\langle D, r, S_1, \ldots, S_n, R_1 \ldots R_p, S^*, \preceq, \mathsf{el}, \mathsf{=} \rangle$ is also a regular automatic tree.

**Proof.**
Let $\langle M_D, M_r, (M'_i)_{1 \leq i \leq n}, (M_i)_{1 \leq i \leq p} \rangle$ be a presentation of $T$. It is enough to describe automata $M_{S^*}$, $M_{\preceq}$, $M_{\mathsf{el}}$, and $M_{\mathsf{=}}$ that run over the canonical encodings of nodes and that capture relations $S^*$, $\preceq$, $\mathsf{el}$, and $\mathsf{=}$, respectively.

- The automaton $M_{S^*}$ for $S^*$ accepts a pair of words $(\alpha, \alpha')$ whenever each of them is accepted by $M_D$ and $\alpha$ is prefix of $\alpha'$.
- The automaton $M_{\preceq}$ for $\preceq$ accepts a pair of words $(\alpha, \alpha')$ whenever each of them is accepted by $M_D$ and the length of $\alpha$ is less than or equal to the length of $\alpha'$.
- The automaton $M_{\mathsf{el}}$ for $\mathsf{el}$ accepts a pair of words $(\alpha, \alpha')$ whenever each of them is accepted by $M_D$ and the length of $\alpha$ is equal to the length of $\alpha'$.
- The automaton $M_{\mathsf{=}}$ for $\mathsf{=}$ accepts a pair of words $(\alpha, \alpha)$ whenever $\alpha$ is accepted by $M_D$.

Restricting to the class of regular automatic trees allows for decidability results of model checking against logics that go beyond $\mathsf{FO}$. 

---

Figure 7.3: An automatic tree that is not regular.
7.3. **Model Checking against $cMSO$ over Regular Automatic Trees**

7.3.3 Model checking against $cMSO$

We now describe the main result of this section, as stated by Theorem 13, where the model checking problem against $cMSO$ is as follows.

**Definition 39** Model checking against $cMSO$

**Input**: The canonical presentation $\langle M_D, M_r, (M_i')_{1 \leq i \leq n}, (M_i)_{1 \leq i \leq p} \rangle$ of a regular automatic tree $T$ and a closed $cMSO$-formula $\varphi$ over the corresponding signature.

**Output**: Yes if $T \models \varphi$, no otherwise.

Theorem 13 can be read as a variant of Theorem 11, where, on the one hand, the logic $FO$ is extended, and, on the other hand, the class of automatic structures is restricted. It can also be read as a generalization of \cite[Th. 5.2]{Tho92}.

**Theorem 13** Model checking against $cMSO$ is decidable over regular automatic trees.

The rest of this section is dedicated to the proof of Theorem 13. This proof goes over the proof made in \cite[Th. 5.2]{Tho92} to generalize the result to arbitrary regular automatic trees.

Before starting the proof, we may assume without loss of generality that all the variables occurring in the formulas are second-order variables. This means to add an extra unary predicate $Sing()$ which conveys that a set is a singleton, and to add inclusion formulas of the form $X \subseteq Y$ so as to capture previous formulas of the form $x \in X$. The reader may refer to \cite{Tho92} for this routine transformation. The syntax then becomes as follows.

$$\varphi := Sing(X) \mid X \subseteq Y \mid R_i(X_1 \ldots X_n) \mid \neg \varphi \mid (\varphi \land \varphi) \mid \exists X \varphi,$$

where $X,Y,X_1, \ldots \in V_2$.

(7.1)

The new syntax requires to adjust the semantics of the formulas, and in particular of those of the form $R_i(X_1, \ldots, X_n)$, by imposing that each $X_j$ is interpreted as a singleton, this is routine. The logic resulting from (7.1) is expressively equivalent to $cMSO$ to the extent that a singleton is a chain. Therefore, in the following, we still use $cMSO$ when using the syntax of (7.1).

A key idea to develop automata construction to model check against $cMSO$ is that a set of addresses is a chain if, and only if, it is contained in the set of all prefixes of some infinite word. Given a chain $C \subseteq \mathbb{E}^*$, we let $Branches(C) := \bigcap\{u\mathbb{E}^* \mid u \in C\}$ be the set of infinite words whose set of prefixes contains $C$. In particular, $Branches(C)$ is a singleton if, and only if, $C$ is infinite.

For $\beta \in Branches(C)$, we write $mbranch(\beta, C)$ for the infinite word over $\mathbb{E} \times \{0, 1\}$ whose first component is $\beta$ and whose second component is marked by 1 if the current prefix of $\beta$ belongs to $C$, and by 0 otherwise. Formally, $mbranch(\beta, C) = m[0]m[1]m[2] \ldots$ with $m[i] = (\beta[i], b)$ where $b = 1$ if $\beta[0] \beta[1] \ldots \beta[i] \in C$, and $b = 0$ otherwise.

The words $mbranch(\beta, C)$ are used to encode the chain $C$. We will overload the encoding identity function $\text{enc}$ of regular automatic trees to denote the encoding of chains. We letting $\text{enc}(C) := \{mbranch(\beta, C) \mid \beta \in Branches(C)\}$. Also, we extend the notation to $\text{enc}(C_1, \ldots, C_m)$ that contains infinite words over alphabet $(\mathbb{E} \times \{0, 1\})^m$ by letting $\text{enc}(C_1, \ldots, C_m) := \text{enc}(C_1) \otimes \ldots \otimes \text{enc}(C_m)$ where $\otimes$ is the convolution extended to sets of infinite words in a natural manner. For $m = 0$, $\text{enc}()$ is the singleton containing the unique infinite word over the singleton alphabet $(\Sigma \times \{0, 1\})^0$.

We now provide an automata-theoretic approach for model checking against $cMSO$ formulas. We first establish the existence of a Büchi automaton $C_m$ that verifies if an infinite word over alphabet $(\mathbb{E} \times \{0, 1\})^m$ denotes a $m$-tuple of chains, as stated by Lemma 4.

**Lemma 4** One can effectively construct a Büchi automaton $C_m$ that recognizes the encoding of $m$-tuples of chains, namely the language $\bigcup_{C_1, \ldots, C_m \in Ch(T)} \text{enc}(C_1, \ldots, C_m)$. 

The automaton $C_m$ of Lemma \ref{lem:factorization} runs $m$ copies of the domain automaton $M_D$. Each copy reads one of the $m$ infinite words over $E$ extracted from the infinite input word over $(E \times \{0, 1\})^m$. Automaton $C_m$ rejects if one of these copies rejects.

Let $\mathcal{T} = \langle D, r, S_1, \ldots, S_n, R_1, \ldots, R_p \rangle$ be a regular automatic tree over the set of directions $E = \{1, \ldots, n\}$ with $T = \langle M_D, \mathcal{M}_r, (\mathcal{M}_i^1)_{1 \leq i \leq n}, (\mathcal{M}_i)_{1 \leq i \leq p} \rangle$ its canonical presentation, and let $\varphi$ be a cMSO-formula\footnote{in the language defined by the grammar \ref{eq:grammar}} with free variables $X_1, \ldots, X_m$. We define the set

$$\text{enc}(\varphi^T) := \bigcup_{\begin{array}{c} C_1, \ldots, C_m \in \text{Ch}(\mathcal{T}) \\mathcal{T}, [X_i \rightarrow C_i]_{1 \leq i \leq m} \models \varphi \end{array}} \text{enc}(C_1, \ldots, C_m) \quad (7.2)$$

**Proposition 12** Given $\varphi$ a cMSO-formula\footnote{with syntax of \ref{eq:grammar}} with free variables $X_1, \ldots, X_m$, one can effectively construct a Büchi automaton $B_{\varphi, T}$ that recognizes the set $\text{enc}(\varphi^T)$.

**Proof.**

We conduct the proof by induction on $\varphi$.

- $\varphi = \text{Sing}(X)$: Automaton $B_{\text{Sing}}$ is obtained by intersecting automaton $C_1$ with an automaton that verifies that the second component of $\text{enc}(C)$ has a single occurrence of symbol 1.

- $\varphi = X \subseteq Y$: Automaton $B_{X \subseteq Y}$ is obtained by intersecting automaton $C_2$ with an automaton that verifies that each time symbol 1 occurs in $\text{enc}(C_1)$ so does it in $\text{enc}(C_2)$.

- $\varphi = R_i(X_1 \ldots X_{r_i})$: Automaton $B_{R_i(X_1 \ldots X_{r_i})}$ is based on $r_i$ copies of automaton $B_{\text{Sing}}$, each one running on a $(E \times \{0, 1\})$-component of the infinite input word over $(E \times \{0, 1\})^{r_i}$, and a variant of automaton $M_i$ for $R_i$. This variant has inputs over $(E \times \{0, 1\})^{r_i}$ instead of $E^{r_i}$ and simulates automaton $M_i$ but replaces component-wise each letter after the unique encountered symbol 1 by the blank symbol $\Box$, so as to mimic the convolution.

- The case $\land$ is a mere intersection, while the case for $\lnot$ requires a complementation (see for instance [Var07]) but then automaton $C_m$ is required again.

- $\exists X \varphi$: It is sufficient to project automaton $B_{\varphi}$. If $\varphi$ has free variables $X = X_1, \ldots, X_m$ this projection corresponds to the mapping $(\Sigma \times \{0, 1\})^m \rightarrow (\Sigma \times \{0, 1\})^{m-1}$. Observe that if $X$ was the only free variable of $\varphi$ we would project onto the unary alphabet.

This achieves the proof of Proposition \ref{prop:factorization}. \hfill \blacksquare
7.4. LOGICS OF KNOWLEDGE AND TIME OVER REGULAR AUTOMATIC TREES

Notice that Proposition 10 from [Bar07] regarding the decidability of the full MSO-theory (and not only cMSO) for automatic presentations based on a singleton alphabet is now a direct corollary of Theorem 13 since every set of words over a singleton alphabet is a chain.

From Theorem 13 on the decidability of cMSO over regular automatic trees, and because binary relations $S^*, \preceq, \text{el} =$ come for free with automata constructions (Lemma 3), we have the following.

**Corollary 2** Model checking against $\text{cMSO}[r, S_1, \ldots, S_n, R_1, \ldots, R_p, S^*, \preceq, \text{el} = ]$ is decidable over a regular automatic trees.

Notice that if we restrict to trees with only relations $S_1 \ldots S_n$, model checking against the full logic MSO becomes decidable (Rabin’s Theorem [Rab69]). However, considering extra relations (e.g. “equal level”) makes the model checking against MSO undecidable over regular automatic trees, as we already saw in Theorem 12 of Section 7.2, which is not the case for cMSO by Corollary 2.

Also, since every path quantifier in a path-MSO-formula can be translated in cMSO, and since considering extra arbitrary relations in trees, such as epistemic relations, does not harm this translation, we obtain the following result, which to our knowledge has not been considered in the literature.

**Corollary 3** Given a regular automatic tree $T = \langle D, r, S_1, \ldots, S_n, R_1, \ldots, R_p \rangle$ and a close path-MSO-formula $\varphi$ over the signature of $T$, it can be decided if $T \models \varphi$.

7.4 Logics of knowledge and time over regular automatic trees

We now exploit the peculiarities of regular automatic trees to consider extensions of several classic formalisms and study their expressivity. We first recall our criterion for expressivity.

**Definition 40** A logic $\mathcal{L}$ is embedded into a logic $\mathcal{L}'$, whenever there is an effective translation that maps every $\mathcal{L}$-formula onto an equivalent $\mathcal{L}'$-formula, namely for every regular automatic tree $T$, we have: $T \models \varphi$ if, and only if, $T \models \varphi'$.

Notice that when $\mathcal{L}$ is embedded into $\mathcal{L}'$, the decidability of the model checking against $\mathcal{L}'$ entails the decidability of the model checking against $\mathcal{L}$.

In tree structures, time is captured by the generalized successor binary relation $S := \bigcup_{i=1}^n S_i$, as in computation trees for temporal logics such as CTL [CE81], CTL* [HT87], etc. Depending on their arities, the other relations $R_1, \ldots, R_p$ have different roles. Unary relations label nodes with atomic propositions that range over some fixed set $AP = \{p, q, \ldots\}$. Binary relations, written $K_1 \ldots K_m$ as the epistemic modalities in multi-agent epistemic logic, will play the role of knowledge modalities as in classic epistemic logic. On this basis, we shortly write $\text{cMSO}K$ for the logic $\text{cMSO}[r, S_1, \ldots, S_n, \langle p \rangle_{p \in AP}, K_1 \ldots K_m]$.

The ability to quantify over chains allows us to capture properties along branches of the trees that are naturally stated in linear-time mu-calculus. Recall that MSO along linear orders, hence branches, is expressively equivalent to Büchi automata that capture all $\omega$-regular properties, as shown by [Tho97], and so does the linear-time mu-calculus [Dam92]. We therefore introduce the logic $\text{BL}_{\mu}^\text{K}$ that is based on the linear-time mu-calculus $\text{BL}_{\mu}$, equipped with knowledge modalities but also with second order quantifications over branches of the tree, just as CTL* extends LTL to the branching-time semantics. The logic $\text{BL}_{\mu}^\text{K}$ is called the branching
epistemic linear-time mu-calculus; in the spirit of CTL*, it relies on two kinds of formulas: state formulas \((\Phi)\) and path formulas \((\varphi)\).

**Definition 41** The syntax of \(BL^{\text{lin}}_K\) is as follows.

- **State formulas:** \(\Phi ::= p | K_i \Phi | \neg \Phi | (\Phi \land \Phi) | E \varphi\) where \(i \in \{1, \ldots, m\}\).
- **Path formulas:** \(\varphi ::= Z | \Phi | \neg \varphi | (\varphi \land \varphi) | X \varphi | \mu Z. \varphi[Z]\) where \(\varphi\) is closed in \(E \varphi\), \(Z \in V\) and \(Z\) is under the scope of an even number of negations in \(\varphi[Z]\).

Formula \(K_i \Phi\) is read as ‘Agent \(i\) knows \(\Phi\).’ Formula \(E \varphi\) is read as ‘there is a path starting from the current state that satisfies \(\varphi\).’ Formula \(X \varphi\) is read as ‘\(\varphi\) holds in the next state.\’ Formula \(\mu Z. \varphi[Z]\) is the linear mu-calculus fix-point construction. Formula \(A \varphi\) is an abbreviation of \(\neg E \neg \varphi\) and means ‘\(\varphi\) holds in all paths starting from the current state.’ Formulas of \(BL^{\text{lin}}_K\) are interpreted in a tree \(T = (D, \tau, S_1, \ldots, S_n, K_1 \ldots K_m, (p)_{p \in AP});\) state formulas are interpreted on nodes, while path formulas are interpreted on branches.

- **State formulas:**
  
  \[
  \begin{align*}
  T, d &\models p & \text{iff} & & d \in p; \\
  T, d &\models K_i \Phi & \text{iff} & & \text{for all } d' \in D \text{ such that } (d, d') \in K_i, \text{ we have } T, d' \models \Phi; \\
  T, d &\models \neg \Phi & \text{iff} & & T, d \not\models \Phi; \\
  T, d &\models (\Phi \land \Psi) & \text{iff} & & T, d \models \Phi \text{ and } T, d \models \Psi; \\
  T, d &\models E \varphi & \text{iff} & & \text{there exists a maximal chain } C \text{ with least element } d
  \end{align*}
  \]

  (that is, a branch/path starting from \(d\))

- **Path formulas:** we add a valuation \(\sigma : V \rightarrow 2^N\) for the free second-order variables in the formulas, and write \(\sigma^{+i}\) for the valuation \(\sigma^{+i}(Z) = \{n | n + i \in \sigma(Z)\}\).

  \[
  \begin{align*}
  T, \pi, \sigma &\models Z & \text{iff} & & 0 \in \sigma(Z) \\
  T, \pi, \sigma &\models \Phi & \text{iff} & & T, \pi(0) \models \Phi; \\
  T, \pi, \sigma &\models (\varphi \land \psi) & \text{iff} & & T, \pi, \sigma \models \varphi \text{ and } T, \pi, \sigma \models \psi; \\
  T, \pi, \sigma &\models \neg \varphi & \text{iff} & & T, \pi, \sigma \not\models \varphi; \\
  T, \pi, \sigma &\models X \varphi & \text{iff} & & T, \pi(1) \varphi(2) \ldots, \sigma^{+1} \models \varphi; \\
  T, \pi, \sigma &\models \mu Z. \varphi[Z] & \text{iff} & & 0 \text{ is in the least-fix point of } [\varphi[Z]]^{\sigma}_{\pi} \text{ defined by: } [\varphi[Z]]^{\sigma}_{\pi} : 2^N \rightarrow 2^N \\
  P \mapsto \{i \in \mathbb{N} | \pi(i) \pi(i + 1) \ldots, (\sigma[Z \mapsto P])^{+i} \models \varphi[Z]\}.
  \end{align*}
  \]

A tree (resp. a forest) satisfies a state formula if it satisfies it from its root node (resp. from all its distinguished root nodes).

Notice that the least fix-point always exists. Indeed, the function \([\varphi[Z]]^{\sigma}_{\pi}\) is monotone (recall that we have required that every variable to occur under the scope of an even number of negations) and the Knaster-Tarski Theorem applies in the complete lattice \((2^N, \subseteq)\). Besides, we can alternatively define the least fix-point of \([\varphi[Z]]^{\sigma}_{\pi}\) as \(\bigcap \{F \subseteq 2^N | [\varphi[Z]]^{\sigma}_{\pi}(F) \subseteq F\}\).

We here give examples of properties that can be expressed in \(BL^{\text{lin}}_K\).

**Example 39**

- \(T, \pi \models \mu Z. p\) if, and only if, \(T, \pi(0) \models p\).
- \(T, \pi \models \mu Z. (XZ \lor p)\) if, and only if, there exists a node \(\pi(i)\) along \(\pi\) such that \(T, \pi(i) \models p\).
- \(T, \pi \models \mu Z. (\psi \lor (\varphi \land XZ))\) if, and only if, there exists a node \(\pi(j)\) such that \(T, \pi(j) \models \psi\) and \(T, \pi(i) \models \varphi\) for every \(0 \leq i < j\).
7.4. LOGICS OF KNOWLEDGE AND TIME OVER REGULAR AUTOMATIC TREES

By means of fix-points in $\mathcal{B}L_{\mu}^\text{lin}K$, we capture the linear-time logic LTL [Pnu77], and since we also have the existential path quantifier $E$, we can embed the full logic CTL* of [HTS77] in $\mathcal{B}L_{\mu}^\text{lin}K$. Now the epistemic modalities $K_i$ allow us to capture CTL*K, the logic CTL* equipped with knowledge modalities. More precisely, we introduce the following macro for $\varphi$ until $\psi$: $\varphi U \psi$ is an abbreviation of formula $\mu \langle \psi \rangle (\varphi \land X \langle \psi \rangle)$.

**Definition 42** $\text{CTL}^\ast K$ is the syntactic fragment of $\mathcal{B}L_{\mu}^\text{lin}K$ defined by the following BNF:

- State formulas: $\Phi ::= \Phi \land \Phi | \neg \Phi | E \varphi | K_i \Phi | p$ where $i \in \{1, \ldots, m\}$.
- Path formulas: $\varphi ::= \neg \varphi | \varphi \land \varphi | X \varphi | \varphi U \varphi$.

**Definition 43** $\text{CTLK}$ is the syntactic fragment of CTL*K defined by the following BNF:

- State formulas: $\Phi ::= \Phi \land \Phi | \neg \Phi | EX \varphi | E(\varphi U \Phi) | A(\varphi U \Phi) | K_i \Phi | p$ where $i \in \{1, \ldots, m\}$.

The following result shows the expressive power of $\mathcal{F}O[K_1 \ldots K_m, (p)_{p \in AP}, S, S^*, \preceq, =]$.

**Proposition 13** $\text{CTLK}$ is embedded into $\mathcal{F}O[K_1 \ldots K_m, (p)_{p \in AP}, S, S^*, \preceq, =]$.

**Proof.**

The key idea is to use relation $S^*$ to consider finite paths and relation $\preceq$ to make these paths as long as we want. We show by structural induction that for all CTLK formula $\Phi$, we can build an FO-formula $f_x(\Phi)$ with one free first-order variable $x$, such that for every tree $(D, r, S_1, \ldots, S_n, R_1, \ldots, R_p)$ and node $d \in D$, the sub-tree $T_d$ of $T$ rooted in $d$ satisfies $\Phi$ if, and only if, $T, [x \mapsto d] \models f_x(\Phi)$. It is not hard to show that the formulas defined below meet this previous condition.

- $f_x(p) := p(x)$; $f_x(\neg \Phi) := \neg f_x(\Phi)$; $f_x(\Phi \land \Psi) := f_x(\Phi) \land f_x(\Psi)$;
- $f_x(K_i \Phi) := \forall y \ K_i(x, y) \to f_y(\Phi)$ where $y$ is a fresh variable;
- $f_x(EX \Phi) := \exists y \ S(x, y) \land f_y(\Phi)$;
- $f_x(E(\Phi U \Psi)) = \exists y \ S^*(x, y) \land f_y(\Psi) \land \forall z \ [(S^*(x, z) \land S^*(z, y) \land y \neq z) \to f_z(\Phi)]$;

The last case for $A(\Phi U \Psi)$ is more involved: notice that if the formula $(\Phi U \Psi)$ holds on every branch of a tree, there must exist a finite depth in the tree such that all the finite branches of the tree up to this depth already witness that $(\Phi U \Psi)$. Indeed, if not, the sub-tree of all branches not satisfying $(\Phi U \Psi)$ would be infinite, and by applying the König lemma to the finite-degree tree would yield a branch not satisfying $(\Phi U \Psi)$, a contradiction. The reciprocal also holds. We therefore let $f_x(A(\Phi U \Psi)) := \exists p \forall t \ (p \preceq t) \to \exists y \ [S^*(x, y) \land S^*(y, t) \land f_y(\Psi) \land \forall z \ [(S^*(x, z) \land S^*(z, y) \land y \neq z) \to f_z(\Phi)]]$.

We end this section with the main Theorem 14 that states the decidability of model checking against $\mathcal{B}L_{\mu}^\text{lin}K$ over regular automatic trees. First of all, it should be clear that the following holds.

**Theorem 14** $\mathcal{B}L_{\mu}^\text{lin}K$ is embedded into $c\text{MSOK}$. 

CHAPTER 7. PROPOSITIONAL EPISTEMIC PLANNING

Proof.

The proof is conducted by induction over the $\mathit{BL}^\mu_K$ formulas. The key idea is to memorize the current node in a first-order $x$ variable, possibly the current branch in a second-order variable $C$, and the variables $Z_1, \ldots$ occurring in the considered $\mathit{BL}^\mu_K$ formula. This proof is rather long, and follows the argument to translate the propositional mu-calculus [Koz83] into MSO. We therefore only explain the case of a formula of the form $\mu Z. \varphi[Z]$ which translates as

$$\exists Z x \in Z \land Z \subseteq X \land (\forall z z \in Z \leftrightarrow \varphi'[z]) \land (\forall Y \forall y y \in Y \leftrightarrow \varphi'[y]) \land Z \subseteq Y,$$

where the formula $\varphi'[z]$ denotes the inductive translation of $\varphi$ (which depends on variable $z$) which expresses that the branch starting from $z$ satisfies $\varphi$. The second conjunct guarantees that $Z$ is the least fix-point.

\[\blacksquare\]

Corollary 4

Model checking against $\mathit{BL}^\mu_K$ is decidable over automatic regular trees.

Figure 7.4 is a recap of the expressivity results we have obtained (arrows resulting from transitivity are omitted), where an arrow $\mathcal{L} \rightarrow \mathcal{L}'$ means that $\mathcal{L}$ is embedded into $\mathcal{L}'$.

![Figure 7.4: Embeddings between FO, cMSO and logics of knowledge and time CTLK, CTL*K, $\mathit{BL}^\mu_K$.](image)

7.5 Application to epistemic planning synthesis

As announced, we show that Theorem 13 together with Proposition 12 have an interesting impact in the domain of epistemic planning. The epistemic planning setting as introduced by [Bol17] relies on Dynamic epistemic logic, that we recall here. Next we explain how the original problem can be generalized to attain more expressive planning goals, such as statements (i)–(ii) of the introduction, while maintaining the decidability frontier for free. Our results generalize the ones obtained in [Man14, AMP14], but make great use of the already established property [Man14, Lemma 22, p. 109] that, under the assumption that preconditions and postconditions of events are propositional, the relational structure that contains all plan candidates is automatic, and actually it is a regular automatic tree (see Theorem 15).

7.5.1 DEL structures

In this subsection, following [Man14], we incorporate the initial epistemic model $\mathcal{M}$ and the infinitely many products $\mathcal{M} \otimes \mathcal{E}^n := \mathcal{M} \otimes \mathcal{E} \otimes \cdots \otimes \mathcal{E}$ into a single structure, called a DEL structure, denoted by $\mathcal{M}\mathcal{E}^*$. Worlds of $\mathcal{M} \otimes \mathcal{E}^n$, called histories, are naturally denoted by words of the form $w e_1 \ldots e_n$ where $w$ is a world of $\mathcal{M}$ and $e_1, \ldots, e_n$ are events of $\mathcal{E}$. E.g. the world $((w, e_1), e_2)$ is denoted by $we_1e_2$. The pair $(\mathcal{M}, \mathcal{E})$ is called the DEL presentation of $\mathcal{M}\mathcal{E}^*$. A DEL presentation $(\mathcal{M}, \mathcal{E})$ is propositional if $\mathcal{E}$ is propositional.

Definition 44 [Man14, Def. 54, p. 105] The DEL structure denoted by $(\mathcal{M}, \mathcal{E})$ is the structure $\mathcal{M}\mathcal{E}^* = (\mathcal{H}, (S_e)_{e \in \mathcal{E}}, (R_a)_{a \in \mathcal{AGT}}, (p)_{p \in \mathcal{AP}})$, where $\mathcal{H}$ is the disjoint union of the sets of worlds of
7.5. APPLICATION TO EPISTEMIC PLANNING SYNTHESIS

\[ M \otimes E^n \] – namely, the histories; for all events \( e \), \( (h, h') \in S_e \) whenever \( h' = he \); \( (h, h') \in R_a \) whenever \( h \) and \( h' \) are worlds of \( M \otimes E^n \), for some \( n \), and \( (h, h') \in R'_a \) where \( R'_a \) is the epistemic relation for agent \( a \) in \( M \otimes E^n \); and finally, \( h \in p \) whenever \( p \in V(h) \) in \( M \otimes E^n \).

Figure 7.5 shows the general shape of a DEL structure generated by the tool Hintikka’s world \[ Sch18 \] available online\[4\]. Black arrows are \( S_e \)-transitions and red and blue arrows correspond to epistemic relations of two distinct agents. The initial epistemic model \( M \) is depicted on the top and is made up of two worlds. The picture then shows \( M \otimes E, M \otimes E^2 \) and \( M \otimes E^3 \). From the proof of in [Mau14, Lemma 22, p. 109], one can easily show that:

**Theorem 15** Propositional DEL structures \( ME^* \) are finite sets (forests) of regular automatic trees, and their presentation is effectively computable from \((M, E)\).

We now turn to the epistemic planning problems.

7.5.2 Generalized epistemic planning and plan synthesis

The epistemic planning problem, as originally stated by [Bol17] is defined as follows:

**Definition 45** Epistemic Planning Problem

- **Input:** a pointed epistemic model \( T, w \), an event model \( E \), an \( \mathcal{L}_{EL} \)-formula \( \psi \);
- **Output:** Yes, if there is a sequence of events \( e_1, \ldots, e_n \) in \( E \) such that \( ME^n, we_1 \ldots e_n \models \psi \).

We generalize this epistemic planning problem by considering a model checking problem against \( BL_{\mu}K \) over DEL structures.

**Definition 46** Generalized Epistemic Planning Problem

- **Input:** a pointed epistemic model \( M, w \), an event model \( E \), an \( BL_{\mu}K \)-formula \( \varphi \);
- **Output:** Yes, if \( ME^* \models \varphi \).

\[ \text{http://hintikkasworld.irisa.fr/} \]
The problem in Definition [16] is indeed a generalization of the epistemic planning problem of Definition [15] in the sense that the latter can be reduced to the former by model checking the formula $\varphi := EF\psi$, where $\psi$ is the planning goal.

The following example shows the relevance of generalized epistemic planning.

**Example 40** We give BL$_\mu$K-formulas $\varphi$ for the three statements (i)–(iii) of the introduction.

1. ‘invariantly, an intruder $a$ does not know the location of the piece of jewelry during more than 3 consecutive steps’ can be expressed as $EG(\Theta \rightarrow (X\neg\Theta \lor X^2\neg\Theta \lor X^3\neg\Theta))$ where $\Theta := \bigvee_{e \in Loc} K_a\text{pieceOfJewelleryIn}(e)$.

2. ‘all drones know that the region is safe every 20 steps’ can be expressed as $E\nu Z.(\bigwedge_{a \in AGT} K_a\text{regionSafe} \land X^{20}Z)$.

3. ‘with the current plan, the drone $a$ never knows that the region is safe but every 10 steps, there is a(nother) plan to let the drone eventually know that the region is safe’ can be expressed as $EG[\neg K_a\text{regionSafe} \land \nu Z.(EFK_a\text{regionSafe} \land X^{10}Z)]$.

The following result subsumes the decidability result for the epistemic planning problem for propositional DEL presentations ([YWL13], [Mau14]), and is a mere corollary of Theorems 15, 14 and 13.

**Theorem 16** The generalized epistemic planning problem is decidable for propositional DEL presentations.

In other terms, in the literature, decidability was established for reachability goals and Theorem 16 says that the problem remains decidable for goals that are arbitrary BL$_\mu$K-formulas $\varphi$. However, when considering planning, only formulas $\varphi$ of the form $E\varphi'$ are relevant. For such formulas, we can effectively build an automaton that recognizes exactly all plans achieving $\varphi'$.

This automaton arises from the automata constructions for BL$_\mu$K-formulas, made possible by the automata constructions for cMSO-formulas, and the fact that every BL$_\mu$K-formula can be effectively translated into a cMSO-formula. The algorithm to obtain this “plan automaton” is as follows.

1. Compute the cMSO-formula $\chi$ equivalent to $\varphi'$ (Th. [14]). By the translation of BL$_\mu$K into cMSO, formula $\chi$ has a single free second-order variable $X$, interpreted as a path, i.e. a plan;

2. Compute $B_\chi$ (Proposition [12]) which accepts all paths achieving goal $\chi$, or equivalently $\varphi'$.

Actually, automaton $B_\chi$ reads infinite words over alphabet $E \times \{0, 1\}$, where the second component of the letters is always equal to 1, since the specified chain is required to be a path. Projecting a word accepted by $B_\chi$ onto the first components of its letters provides a plans.

**7.6 Discussion**

We have adapted the seminal result of the decidability of the FO-theory of every automatic structure by augmenting the logic up to cMSO, a logic relying on chain-MSO of [HTS7, Tho87], but by restricting to the subclass of regular automatic trees. A nice application of this result is the decidability of the generalized epistemic planning problem for propositional DEL presentations.
Regarding the computational complexity of our algorithms, it should be observed that the automata construction to model check against cMSO (Proposition 12) is non elementary in the number of alternations between existential and universal quantifiers in the formula (each underlying negation yields an exponentiation blow-up for the complementation). Still, it would be relevant to investigate in practice, if the automata have particular shapes that do not reach this non-elementary worst-case upper-bound complexity.

Compared to the work of Maubert [Mau14] with CTL∗K, we offer the entire logic cMSO (or BLlinµK) that strictly subsumes CTL∗K, and yet allowing to solve the epistemic planning problems over the same class of planning domains (i.e. propositional DEL presentations). However, it should be noticed that [Mau14] also considers a different problem called the epistemic protocol synthesis problem, which differs from the generalized epistemic planning problem: in the epistemic protocol synthesis problem, one has to prune the tree structure so that the remaining satisfies a CTL∗K-goal. This subject is out of the scope of this paper, but suggests some comments.

The existence of a pruning, i.e. a sub-tree, that satisfies the protocol goal, requires the use of second-order quantifiers ranging over arbitrary subsets of nodes, that is the full logic MSO which cannot be model checked (Theorem 12). It is a open question whether there is a logic in between cMSO and full MSO to solve the epistemic protocol synthesis problem with BLlinµK specifications. The existence of a pruning of the tree that satisfies the protocol goal requires the use of second-order quantifiers ranging over arbitrary subsets of nodes, which is the full logic MSO that cannot be model checked (Theorem 12). It is an open question whether or not there is room for a logic in between cMSO and full MSO to solve the epistemic protocol synthesis problem with BLlinµK specifications. Also, the synthesis technique deployed by [Mau14] actually applies to a family of tree structures that is larger than the one of regular automatic trees. At the moment, the technique does not adapt to logics that involve arbitrary fix-points. It is an open question whether or not the work of [Mau14] can be faithfully extended to handle the entire alternation-free fragment of BLlinµK.
Part III

Applicability in AI
Chapter 8

Seeing and knowing

Figure 8.1: NASA’s Mars Exploration Rover.

In many practical settings requiring higher-order knowledge (video games, multi-UAV systems, multi-robot systems, robot interacting with humans, etc.), autonomous agents are located in a geometrical environment. For these applications, the approach advocated by Halpern and Vardi is to use model checking for reasoning about knowledge [HV91]. Thus, epistemic models are needed to give semantics to epistemic formulas. On the other hand, epistemic models are not directly available. Indeed, the information agents get from the environment is low level and comes from sensors and is about the world itself, for instance positions of agents.

In this chapter, we attempt to bridge the gap between sensor information and epistemic models. More precisely, we attempt to infer an epistemic model from geometrical information. Our proof-of-concept is simple: sensors of agent \( a \) output the set of agents seen by \( a \).

8.1 Inferring epistemic models

In this section, we consider a set \( S \) of points in the geometrical environment. A possible world, called an \( S \)-world consists of positions of agents and a visibility function as given in the following definition. In other words, a possible world is a structured mathematical object that describes the position and the view of all agents.

**Definition 47** Let \( S \) be a non-empty set. An \( S \)-world is a pair \((pos, vis)\) where:

- \( pos : AGT \to S \);
- \( vis : AGT \to 2^S \) such that for all agents \( a \in AGT \), \( pos(a) \in vis(a) \).
In Definition 47, the function \( pos \) assigns a position in \( S \) to each agent in \( AGT \). The function \( vis \) maps each agent \( a \) to the set of points \( \text{vis}(a) \) seen by \( a \). We suppose that an agent sees its own location. Now, we define the corresponding epistemic model of a given class \( C \) of \( S \)-worlds. Furthermore, we only consider structured atomic propositions of the form \( a \triangleright b \), where \( a, b \) are agents. The informal reading of \( a \triangleright b \) is ‘agent \( a \) sees agent \( b \).

**Definition 48** Given a class \( C \) of \( S \)-worlds, we define the epistemic model \( M_C = (W, (R_a)_{a \in AGT}, V) \) where:

1. \( W = C \);
2. \( R_a \) over \( W \) is defined by:
   \[
   (\text{pos}, \text{vis})R_a(\text{pos}', \text{vis}') \text{ iff for all agents } b, \text{ if } \text{pos}(b) \in \text{vis}(a) \text{ then } \text{pos}(b) = \text{pos}'(b) \text{ and } \text{vis}(b) = \text{vis}'(b);
   \]
3. \( V(\langle \text{pos}, \text{vis} \rangle) = \{ a \triangleright b \mid \text{pos}(b) \in \text{vis}(a) \} \).

Item 2 of Definition 48 defines the epistemic relation \( R_a \) in an intuitive way: two \( S \)-worlds are indistinguishable for an agent \( a \) if agent \( a \) sees the same thing in both worlds. More precisely, for all agents \( b \) seen by \( a \), agent \( b \) has the same positions and the same vision in both \( S \)-worlds. Item 2 defines the epistemic relation \( R_a \) globally, meaning that there is common knowledge of the ability of agents to sense other agents and that all agents have the same ability. Item 3 says that proposition \( a \triangleright b \) holds in a given world iff agent \( b \) is seen by agent \( a \).

Let \( C \) be a class of \( S \)-worlds. We consider the following model checking problem, called the \( C \)-model checking problem:

- **input:** the description of a \( S \)-world \( w \), an epistemic formula \( \varphi \);
- **output:** yes if \( M_C, w \models \varphi \); no otherwise.

Several classes \( C \) have been considered in the literature. The two next sections review them.

### 8.2 Agents than can move

Given an integer \( d \) and an angle \( \alpha \), Balbiani et al. [BGS13] consider the class \( C^\alpha_d \) of \( \mathbb{R}^d \)-worlds \( \langle \text{pos}, \text{vis} \rangle \) where \( \text{vis}(a) \) is a cone of angle \( \alpha \) centered in \( \text{pos}(a) \) (actually, in [BGS13], the angle \( \alpha \) is assumed to be equal to \( \pi \) so that agents see closed hyperplanes, but that assumption is not really important). Two indistinguishable \( C^\alpha_d \)-worlds for agent \( a \), with \( \alpha = \frac{\pi}{5} \), are given in Figure 8.2. For the one-dimensional case, we omit the parameter \( \alpha \) since angles are irrelevant. Figure 8.3 shows two indistinguishable \( C^\alpha_1 \)-worlds for agent \( a \).

The \( C^\alpha_d \)-model checking is decidable, by reduction to the satisfiability of a first order formula over the theory of the real numbers, known to be decidable [BGS13]. It is \( \text{PSPACE} \)-hard. The exact complexity of the model checking problem is still open, and is quite challenging. The main issue is the following: if \( d \geq 2 \), the model \( M_{C^\alpha_d / \sim} \) defined as the quotient of \( M_{C^\alpha_d} \) and the equivalence relation \( \sim \), defined by \( w \sim u \) if the same atomic propositions of the form \( a \triangleright b \) hold in both \( w \) and \( u \) is not bisimilar to \( M_{C^\alpha_2} \). Let us give an example (actually a variant of the example given in [BGS13] and [Sch10]) that explains that issue.

**Example 41** Figure 8.4 actually two \( C^\alpha_2 / \sim \)-worlds \( w \) and \( w' \) that satisfy exactly the same atomic formulas of the form \( \cdot \triangleright \cdot \) (agents \( b, c, d, e \) see no other agents and \( a \) sees \( b, c \) and \( d \)), but...
8.2. AGENTS THAN CAN MOVE

Figure 8.2: Two indistinguishable $C_2^\alpha$-worlds for agent $a$ with $\alpha = \frac{\pi}{3}$.

Figure 8.3: Two indistinguishable $C_1$-worlds for agent $a$.

$K_a((e \triangleright b \land e \triangleright d) \rightarrow e \triangleright c)$ holds in $w$ and not in $w'$. Indeed, for $w$, as agent $a$ sees that $c$ is in the segment $bd$, if $b, d$ are in the cone of vision of $e$ and then $c$ is also, and therefore, $K_a((e \triangleright b \land e \triangleright d) \rightarrow c \triangleright c)$ in $w$. However in $w'$, world $u'$ depicted Figure 8.5 is possible for agent $a$ and does not satisfy $(e \triangleright b \land e \triangleright d) \rightarrow e \triangleright c)$. Therefore, $K_a((e \triangleright b \land e \triangleright d) \rightarrow e \triangleright c)$ does not hold in world $w'$.

In other words, there is no strong evidence that there exists a finite model equivalent to $M_{C_2^d}$ with respect to a finite set of agents, if $d \geq 2$. We do not know whether a finite axiomatization for the validities over $M_{C_2^d}$ exists.

However, the one-dimensional case is easier. It is known that the $C_1$-model checking problem is in $\text{PSPACE}$-complete. If $d = 1$, we can make use of a symbolic representation. More precisely, $M_{C_1}/\sim$ is bisimilar to $M_{C_1}$ [BGS13]. As model $M_{C_1}/\sim$ can be represented by accessibility relations (see Chapter 3), $C_1$-model checking is in $\text{PSPACE}$ (and also $\text{PSPACE}$-hard).

Figure 8.4: Two $C_2^{\pi/6}$-worlds that satisfy the same atomic propositions, but that are modally non equivalent.
8.3 Rotating cameras

Gasquet et al. [GGS15] considered another setting where positions of agents are fixed and common knowledge among all the agents but not the directions they are looking at. In other words, the setting consists of rotating cameras. Gasquet et al. only consider the two-dimensional case. Formally, it consists of fixing a given \( pos' : AGT \rightarrow \mathbb{R}^2 \) and a given angle \( \alpha \), and considering the class \( C_{2}^{\alpha, pos'} \) of \( \mathbb{R}^2 \)-worlds \( \langle pos, vis \rangle \) where \( vis(a) \) is a cone of angle \( \alpha \) centered in \( pos(a) \) and \( pos' = pos \). In that setting, agents have the same positions in all worlds, just the direction of views may be different. Figure 8.6 shows two indistinguishable worlds for agent \( a \) in that settings.

Interestingly, when the positions of agents are common knowledge, there is a finite representation and the \( C_{2}^{\alpha, pos'} \)-model checking is in \( \text{PSPACE} \)-complete [GGS15]. It is essentially due to the fact a symbolic representation exists: models \( M_{C_{2}^{\alpha, pos'}} \) and \( M_{C_{2}^{\alpha, pos'}/\sim} \) are bisimilar, where \( \sim \) is the equivalence relation ‘same truth values for the atomic propositions of the form \( a \triangleleft b \).

8.4 Geometry in the syntax: poor’s man logic

Herzig et al. [HLM15] have proposed a language where geometrical constraints are written in the language and are not grounded in the models – only the semantics of the knowledge operator.
is still grounded. Namely, in the framework, we can express observability constraints such as ‘an agent $a$ sees that an agent $b$ sees the truth value of $p$’ (written $S_a S_b p$). In [CHL+16], this logic is extended with public announcement operators. It is proven that the model checking problem is PSPACE-complete, the model being a valuation over observability constraints.

### 8.5 Perspectives

It is natural to extend epistemic logic on geometrical environments with actions. If the environment is discrete and synchronous, event models are perfectly suitable to capture actions, such as private announcement of $p$ by agent $a$ to all agents that see agent $a$.

Interestingly, a long avenue of research would be to design a framework where actions are continuous in time. A promising approach could be to mix event models with differential equations and to build hybrid models [Pla18].
Chapter 9
Asynchronous systems

This chapter is adapted from [KMS17].

9.1 Introduction

Asynchrony plays a central role in distributed systems such as robotic rescue teams, smart cities, autonomous vehicles, etc. In such systems, there may be an unpredictable delay between sending and receiving messages, and there is not always access to a centralized clock. Recently, with the proliferation of multi-agent systems (MAS) where independent agents interact, communicate, and make decisions under imperfect information, modeling the evolution of knowledge as informative events occur has become increasingly important for both verification and design. For instance, it is often crucial to know whether an agent has received some information, so it would be highly desirable to be able to analyze messages such as “agent $a$ knows that agent $b$ received message $m$”, i.e., we want to model

$$\text{messages with epistemic content,}$$

and because we are considering automated systems where agents do not lie, make logical mistakes, or have inaccurate factual information (for instance autonomous vehicles communicating about their position), we also make the classic assumption that

$$\text{messages are true when they are sent.}$$

Public announcement logic (PAL) [Pla07] is one of the first and most influential proposals for modelling the relationship between knowledge and announcements. In PAL, true announcements are made to a group of agents. This logic later led to the powerful and much studied dynamic epistemic logic (DEL) [Van07], which can describe more complex forms of communication, such as semi-private announcements, private announcements, and much more. However, both these logics assume synchronicity: In PAL messages are immediately received by all agents at the same time, as soon as they are sent, and in DEL agents may perceive events differently, but events immediately change the epistemic state of agents as soon as they occur. In asynchronous settings, however, messages are not delivered instantly, and agents may receive them at different points in time, making PAL intrinsically unfit for reasoning about such settings. This fact becomes even more evident when we consider that in PAL, every announcement immediately leads to common knowledge, while common knowledge is not achievable in asynchronous systems.
The only work we know of considering how DEL can capture asynchrony is [DLW11], but in this logic an agent can only consider possible “future” events if they do not change her epistemic state. This is related to the principle of inertia [vL10, BBP16], which states that in absence of any observation, one assumes that nothing has happened. This assumption is natural in contexts where agents believe that they can observe all events at the time of their occurrence. In asynchronous systems however, agents should know that even when they do not observe anything, or when they do not receive any messages, it is possible that messages are being sent and received by other agents. Therefore the inertia principle does not apply in our setting, and agents should be able to imagine possible pending messages.

Our aim is to propose a logic in the spirit of PAL for reasoning about (9.1) epistemic messages in asynchronous systems, (9.2) that are true at the time of announcement, and where (9.3) agents can imagine messages that have been sent but not yet received.

Because this is an ambitious endeavor, we make a few assumptions to start as simply as possible:

- **Broadcasts**: all messages are sent to every agent
- **External source**: messages are emitted by an external, omniscient source
- **FIFO**: messages are received in the order they are sent.

The first assumption comes from PAL, and is natural in the context of smart cities for instance, where autonomous vehicles broadcast their current position or direction. The second one is a choice made to simplify the syntax by not having to model the origin of announcements, as in PAL. Announcements from an external, omniscient source can in some cases be used to model messages broadcast by agents within the system, in particular, an omniscient outside agent broadcasting that agent $a$ knows $\varphi$ is in many situations equivalent to agent $a$ broadcasting $\varphi$ to the other agents within the system. This captures the fact that in order for an agent within the system to make a true announcement, she should know that the announcement is true before she broadcasts it. Thus upon receiving an announcement made by agent $a$, another agent learns both the announcement and the fact that $a$ knows it, modeled within the framework where announcements come from an external source by the announcement of $K_a \psi$ ($K_a$ being the classic knowledge operator of epistemic logics, see for instance [FMHV03a]). However, depending on assumptions about the agents’ epistemic and reasoning capacities, this way of modeling agents’ announcements may not be completely faithful to the real situation. We discuss this issue briefly in the future work section.

Concerning the last assumption, FIFO is a simple but classic scheme of communication in asynchronous systems (see for instance [YG82, BZ83, CS08]).

Figure 9.1 depicts the architecture of such a system with three autonomous agents receiving messages from a public channel and reading them when they are ready to process them. To represent the fact that agents read messages in the order they were sent, we provide each one with a private FIFO channel. Each copy receives the same messages from the public channel, in the order in which they are announced, but the moment at which these messages are read differs from one agent to another.

In PAL, messages are received at the same time they are sent, and thus the announcement operator combines both sending and receiving. In contrast, in our setting, messages are not received immediately and they may be received at different times by different agents. The syntax reflects this aspect by providing both a sending operator, which adds new messages to the public channel, and a receiving operator for each agent, which allows her to read the first message in her FIFO queue that she has not read yet. Thus, in our logic, we provide the following modal constructions:
9.1. INTRODUCTION

Figure 9.1: Agent architecture

- \( \langle \psi \rangle_{\text{async}} \phi \), which means “\( \psi \) is currently true, and after its announcement, \( \phi \) holds”;
- \( \Box_{a} \phi \), which means “after agent \( a \) reads the next announcement, \( \phi \) holds”;
- \( K_{a} \phi \), which means “agent \( a \) knows that \( \phi \) holds”.

Interestingly, the natural semantics for this logic presents a challenging problem of circular definition: In order to define the truth of epistemic formulas, we classically quantify over the set of all states that the agent considers possible, where states include the current content of the public channel and pointers to the last message read by each agent. But some states are not consistent and must not be considered: intuitively a state is consistent if the announcements it contains were true at the time they were made. Therefore, defining consistent states requires defining the truth of formulas, and vice versa. PAL presents a similar circularity, as the definition of the update of a model by an announcement and the definition of the truth values are mutually dependent, thus this phenomenon is not inherent to the asynchronous setting. However, only asynchrony makes it a problem. Indeed in PAL a simple definition by mutual induction is possible. In an asynchronous setting, however, agents do not know what or how many messages other agents have received; in particular, an agent may consider it possible that some formula has been announced that is bigger than all those that have actually been announced, which makes a mutual induction impossible for lack of a decreasing quantity. This circularity problem is inherent in the asynchronous setting, and is independent from the assumptions of broadcasts, external source and FIFO described above. It only depends on the assumptions that announcements can talk about knowledge (9.1), that they must be true (9.2), and that agents can imagine pending messages (9.3).

We partially solve the issue by defining three restricted cases in which we manage to avoid circularity. The first one requires the Kripke model representing the initial epistemic situation to be a finite tree; the second one only allows announcements from the existential fragment of our logic, and the third one makes the assumption that only a bounded number of announcements can be made during each time unit (a strong form of non-Zeno assumption), and that agents have access to a global clock. In the second case, the semantics is defined thanks to an application of the Knaster-Tarski fixed point theorem [Tar55].

We then discuss some properties of our logics, compare them to PAL, and establish some validities that hold whenever the semantics is well-defined; we also study the model checking problem for our logic and establish the following complexity results:

<table>
<thead>
<tr>
<th>Restrictions</th>
<th>Complexity of model checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional announcements</td>
<td>( \text{PSPACE-complete} )</td>
</tr>
<tr>
<td>Finite tree initial models</td>
<td>( \text{in PSPACE} )</td>
</tr>
<tr>
<td>Announcements from the existential fragment</td>
<td>( \text{in \text{EXPTIME}, PSPACE-hard} )</td>
</tr>
</tbody>
</table>
Finally, we study the satisfiability problem in the case of propositional announcements, and we establish that it is $\textsf{NExpTime}$-complete.

The chapter is organized as follows. In Section 9.2 we present our logic, and discuss the circularity problem that arises from the definition of the semantics, and present an example to illustrate the logic. In Section 9.3 we exhibit cases where it can be solved. We then present some properties in Section 9.4 (a comparison with PAL and some validities), we study the model checking problem in Section 9.5 and the satisfiability problem in Section 9.6. Finally we discuss related work in Section 9.7 and future work in Section 9.8.

This chapter is a summary of [KMS15] and [KMS17].

9.2 Settings

In this section we present our framework for reasoning about asynchronous epistemic announcements in a public channel. As in Chapter 2, $\mathcal{AP}$ is a countable infinite set of atomic propositions, and $\mathcal{AGT}$ is a finite set of agents. For pedagogical reasons, we first introduce models, then the syntax and finally the semantics of our logic, even though by doing so we need to refer to the language before we formally define it.

9.2.1 Models

Agents start with an initial state of knowledge of the world, which is modeled by an initial pointed epistemic model, or Kripke model. Then true announcements are made by some external entity, and sent in the public channel. The whole sequence of announcements that have been made up to the present moment is modeled as a sequence of formulas from our logic, whose syntax we introduce later in Section 9.2.2. Agents read these messages independently, possibly at different times, but in FIFO order. To represent which messages each agent has already read, and thus which ones remain to be read, we simply map each agent to the number of announcements she has read. Such a mapping is called a cut.

Initial Kripke model

An initial model is given as a Kripke model $\mathcal{M} = (W, \{R_a\}_{a \in \mathcal{AGT}}, \Pi)$, as defined in Definition 1. It represents the initial knowledge of agents before any announcements are made, and it corresponds to the notion of initial knowledge in [Ray13], p. 5. In practice, $(\mathcal{M}, w)$ is directly provided by the modeller or inferred from what agents perceive [BGS13, GGS15].

Announcements

We consider that, in a given scenario, not every formula may be announced, but rather that there is a certain set of relevant announcements. Furthermore, we allow the number of times an announcement can be made to be bounded. To represent this, we define the notion of announcement protocols ($L_{async}$ is the language defined in Section 9.2.2).

Definition 49 An announcement protocol is a multiset of formulas in $L_{async}$, where the multiplicity of an element $\psi$ is either an integer or $\infty$.

Example 42 The reader may imagine a card game where it is only possible to announce ‘agent $a$ has a heart card’ once and ‘agent $a$ does not know whether agent $b$ has a heart card or not’ twice. We let the proposition $\Diamond_a$ mean “agent $a$ has a heart card,” and define the announcement protocol to be $\{(\Diamond_a, K_a \Diamond_b \land K_a \neg \Diamond_b), K_a \Diamond_b \land K_a \neg \Diamond_b\}$. 


9.2. SETTINGS

Given an announcement protocol $\mathcal{A}$, we denote by $\text{Seq}(\mathcal{A})$ the set of finite sequences $\sigma = [\varphi_1, \ldots, \varphi_k]$ such that the multiset $\{{\varphi_1, \ldots, \varphi_k}\}$ is a submultiset of $\mathcal{A}$. We define the size of a sequence $\sigma$ as $|\sigma| := \sum_{i=1}^{k} |\varphi_i|$. For $\sigma, \sigma' \in \text{Seq}(\mathcal{A})$, we write $\sigma \leq \sigma'$ if $\sigma$ is a prefix of $\sigma'$. The sequence $\sigma|_k$ is the prefix of $\sigma$ of length $k$. Given a formula $\varphi$ and a sequence of formulas $\sigma$, $\varphi ::= \sigma$ (resp. $\sigma ::= \varphi$) is the sequence obtained by adding $\varphi$ at the beginning (resp. at the end) of $\sigma$.

States

We now define the set of possible states of the models in which the formulas of our logic will be evaluated.

**Definition 50** Let $\mathcal{M}$ be an initial model and $\mathcal{A}$ an announcement protocol. We define the set of possible states $\mathcal{S}_{\mathcal{M}, \mathcal{A}}$ as follows:

$$\mathcal{S}_{\mathcal{M}, \mathcal{A}} = \{(w, \sigma, c) \mid w \in W, \sigma \in \text{Seq}(\mathcal{A}) \text{ and } c : AGT \rightarrow \{0, \ldots, |\sigma|\}\}.$$  

The first element of a state represents the world the system is initially. The second element is the list of messages that have already been announced. The last element, $c$, is called a cut, and for each $a \in AGT$, $c(a)$ is the number of announcements of $\sigma$ that agent $a$ has received so far. Given two cuts $c$ and $c'$, we write $c < c'$ if for all $a$, $c(a) \leq c'(a)$ and there exists $b$ such that $c(b) < c'(b)$; in other words, $c < c'$ if all agents have received at least as many messages in $c'$ as in $c$, and at least one agent received strictly more messages in $c'$. Typical elements of $\mathcal{S}_{\mathcal{M}, \mathcal{A}}$ are denoted $S, S'$, etc.

**Example 43** Consider the state $S = (w, [\varphi, \psi, \chi], c)$ where $c(a) = 2$ and $c(b) = 1$. $S$ represents the situation where in initial world $w$, the sequence $[\varphi, \psi, \chi]$ of formulas has been announced, agent $a$ has received $\varphi$ and $\psi$, and agent $b$ has only received $\varphi$. Only $\chi$ remains in the queue of $a$ and has not been read yet, and only $\psi$ and $\chi$ remain in the queue of $b$. We represent $S$ as follows:

and we may also write $S = (w, [\varphi, \psi, \chi], a \mapsto 2, b \mapsto 1)$.

**Example 44** Consider the state $S = (w, \epsilon, 0)$, where $\epsilon$ denotes the empty sequence of formulas and $0$ is the function that assigns $0$ to all agents. $S$ represents an initial world $w$ in which no announcement has been made (and therefore no announcement has been received either). It can be represented as follows:

**Example 45** State $S = (w, [\varphi, \chi], 0)$, which represents the situation where in initial world $w$, $\varphi$ and $\chi$ have successively been announced, but neither agent $a$ or agent $b$ received any announcement yet. We depict it as follows:

$$w a : \varphi \chi \quad b : \text{no messages}$$
Consistent states

Definition 50 allows for all combinations of worlds, sequences of announcements allowed by the announcement protocol, and cuts. This definition is an over-approximation of the set of states we want to consider: indeed, because announcements must be true, some of the states in $S$ are inconsistent. For example, suppose that $w$ is a world in $M$ where $p$ does not hold. Because only true announcements can be made, $p$ cannot be announced in world $w$, and thus the state $(w, [p], 0)$ is inconsistent.

Example 46 Let us consider the following initial model, where $w, u, v, z$ are worlds, $a$ and $b$ are agents and $p$ is a proposition. The arrows represent the agents’ accessibility relations, before any announcements have been made. So at world $w$, agent $a$ considers $u$ and $v$ possible, and agent $b$ considers world $z$ possible.

Now assuming that the announcement protocol $A$ contains $p, \varphi$ and $\psi$, a partial depiction of the asynchronous model $M \otimes A$ is below. We depict the states $w, u, v, z$ where no announcement has been made, as well as copies of $u$ where two different sequences of announcements have been made, and received in one state by agent $b$ and in a different state by agent $a$. Of course, the entire model $M \otimes A$ is infinite so we do not depict all the states here.

State $\begin{array}{c} u \\ a \\ b \end{array} \begin{array}{c} p \\ \varphi \end{array}$ is not consistent because $p$ has been announced even though $p$ is not true in $u$. This notion of inconsistency is the source of the circularity problem, as we discuss in Section 9.2.4. For now, we define the relations that capture which states an agent considers possible before removal of inconsistent ones.

Pre-accessibility relation and asynchronous pre-model

We now define, for each agent, a pre-accessibility relation that does not yet take consistency into account, but is only based on the agents’ accessibility relations in the initial model, and the messages it has already read.

Definition 51 The pre-accessibility relation for agent $a$, denoted by $R_a$, is defined as follows: given $S = (w, \sigma, c)$ and $S' = (w', \sigma', c')$, we have $SR_aS'$ if:

1. $wR_aw'$, and
2. $\sigma|_{c(a)} = \sigma'|_{c'(a)}$. 
9.2. SETTINGS

The first clause simply says that for $S'$ to be considered possible by $a$ when in $S$, world $w'$ must be considered possible by $a$ from $w$. The second clause says that agent $a$ is aware of, and aware only of, messages that she has received: therefore she can only consider possible states where she has received exactly the same messages. Then, because the principle of inertia does not apply to the asynchronous setting, she can imagine any possible sequence of pending announcements, as long as it is compatible with the announcement protocol. Also, as she has no information about what messages the other agents have received, $c'(b)$ can be anything if $b \neq a$: agent $a$ considers it possible that $b$ received more, or fewer, messages than she actually has. Note that the second clause also implies $c(a) = c'(a)$.

Given an initial model $M$ and an announcement protocol $A$, we define the asynchronous pre-model $M \otimes A := (S_{M,A}, \{R_a\}_{a \in AGT})$, where $S_{M,A}$ is the set of possible states, and for $a \in AGT$, $R_a$ is the pre-accessibility relation for agent $a$.

9.2.2 Language

We now introduce the syntax of our logic, which we call Asynchronous Broadcast Logic, or $L_{async}$ for short. Note that we do not use the term “public announcement” in the name of our logic as it has a strong synchronous connotation: public announcements are often thought of as becoming common knowledge the moment they are made.

Definition 52 (Syntax) The set of $L_{async}$-formulas is given by the following grammar:

$$\varphi ::= p \mid (\varphi \land \psi) \mid \neg \varphi \mid K_a\varphi \mid \langle \psi \rangle_{async} \varphi \mid \bigcirc_a \varphi,$$

where $p$ ranges over $AP$ and $a$ ranges over $AGT$.

The intuitive meaning of the last three operators is the following: $K_a\varphi$ means that agent $a$ knows $\varphi$, $\langle \psi \rangle_{async} \varphi$ means that $\psi$ is true and after $\psi$ has been put on the public channel, $\varphi$ holds, and $\bigcirc_a \varphi$ means that agent $a$ has a pending message, and after she has received and read it, $\varphi$ holds. We define the dual of the announcement operator: $[\psi]\varphi := \neg \langle \psi \rangle_{async} \neg \varphi$, meaning that if $\psi$ is true, then $\varphi$ holds after its announcement. $|\varphi|$ is the length of $\varphi$, and we denote by propositional formula a formula that uses no modalities, i.e., contains no occurrences of $K_a$, $\langle \psi \rangle_{async}$, or $\bigcirc_a$.

In (synchronous) public announcement logic (see Definition 9), the operator $\langle \psi \rangle$ captures both the broadcast and the reception of an announcement $\psi$, because in the synchronous setting, sending and reception occur simultaneously. In our asynchronous setting, not only can sending and reception occur at different times, but also different agents may receive the same message at different times. Therefore we capture the broadcast of a formula $\psi$ with operator $\langle \psi \rangle_{async}$, while agent $a$’s reception of a broadcasted formula is captured by the operator $\bigcirc_a$.

9.2.3 Truth conditions

For the rest of the section, we fix an initial model $M$ and an announcement protocol $A$. As discussed in Section 9.2.1, some possible states from Definition 50 are inconsistent, because they contain announcements that were not true at the time they were announced. Also, because agents should not consider inconsistent states possible, we described how defining consistency is necessary to define the semantics of the knowledge operator, which in turn is necessary to define the consistency of states that contain epistemic announcements, hence a circularity problem.

We describe in Figure 9.2 the definition of consistency (represented with symbol ✓) as well as truth conditions for our logic. This definition is circular, and therefore the semantics as presented here is not well-founded, although it conveys the intended meaning of our operators. In the next section we will describe restricted cases in which we can provide a semantics that is well-defined.
CHAPTER 9. ASYNCHRONOUS SYSTEMS

Truth conditions for consistency:

\[(w, \epsilon, 0) \models \checkmark\] (always)
\[(w, \sigma, c) \models \checkmark\] if there is \(c' < c\) s.t. \((w, \sigma, c') \models \checkmark\), or
\[
\sigma = \sigma' \triangleright \psi; (w, \sigma', c) \in S_{M,A},
\]
\[(w, \sigma', c) \models \checkmark\] and \((w, \sigma', c) \models \psi\)

Truth conditions for formulas:

\[(w, \sigma, c) \models p\] if \(p \in \Pi(w)\)
\[(w, \sigma, c) \models (\varphi_1 \land \varphi_2)\] if \((w, \sigma, c) \models \varphi_1\) and \((w, \sigma, c) \models \varphi_2\)
\[(w, \sigma, c) \models \neg \varphi\] if \((w, \sigma, c) \not\models \varphi\)
\[(w, \sigma, c) \models K_a \varphi\] if for all \(S'\) s.t. \((w, \sigma, c) R_a S'\) and \(S' \models \checkmark, S' \models \varphi\)
\[(w, \sigma, c) \models \langle \psi \rangle_{\text{async}} \varphi\] if \(\sigma ; \psi \in \text{Seq}(A)\), \((w, \sigma, c) \models \psi\) and \((w, \sigma ; \psi, c) \models \varphi\)
\[(w, \sigma, c) \models \Box_a \varphi\] if \(c(a) < |\sigma|\) and \((w, \sigma, c^{+a}) \models \varphi\)

where \(c^{+a}(b) = \begin{cases} c(b) & \text{if } b \neq a \\ c(b) + 1 & \text{if } b = a \end{cases}\)

Figure 9.2: Consistency and semantics

The intuitive meaning of \((w, \sigma, c) \models \checkmark\) is that the state \((w, \sigma, c)\) is consistent, that is, all announcements in \(\sigma\) were true when they were made. The first clause is obvious: the initial state where no announcement has been made is consistent. The second clause gives two possibilities for a state to be consistent. Either there was an earlier consistent state \((w, \sigma, c')\) in which some agents received some already announced formulas, increasing the cut from \(c'\) to \(c\), or a new, true announcement \(\psi\) has been made from an earlier consistent state, extending the history from \(c'\) to \(c' : \psi\).

For the formulas, the first three clauses are straightforward. The fourth clause says that agent \(a\) knows \(\varphi\) if \(\varphi\) holds in all consistent states that she considers possible. The fifth clause says that \(\langle \psi \rangle_{\text{async}} \varphi\) holds in a state \(S\) if \(\psi\) can be announced (it is true in \(S\)), and \(\varphi\) holds in the state obtained by adding \(\psi\) to the public channel. The last clause says that \(\Box_a \varphi\) holds if agent \(a\) has at least one unread announcement in the channel, and \(\varphi\) holds after she reads the first unread message.

9.2.4 Circularity

By observing the truth conditions for consistency and for formulas in Figure 9.2, one can see that defining whether a state is consistent requires one to define whether an announcement can be made, and this requires the semantics of our logic to be defined. But to define the semantics of the knowledge operators, we need to define which consistent states are considered possible by the agent, which requires us to define which states are consistent, hence the circularity.

Let us consider the following example, where \(AGT = \{a\}\). Let the initial model be \(M = (W, R_a, \omega)\) where \(W = \{w\}\), \(R_a = \{(w, w)\}\) and \(\omega(w) = \emptyset\), and let the announcement protocol be \(A = \{\{K_a p\}\}\). According to Figure 9.2 we have: \((w, [K_a p], 0) \models \checkmark\) iff \((w, \epsilon, 0) \models K_a p\). But, as \((w, \epsilon, 0) R_a (w, [K_a p], 0)\), the definition of the truth value of \((w, \epsilon, 0) \models K_a p\) depends on the truth value of \((w, [K_a p], 0) \models \checkmark\). To sum up, the definition of \((w, [K_a p], 0) \models \checkmark\) depends on itself.

The circularity problem depends on assumptions (1), (2), and (3) from the introduction:

- Announcements can be epistemic
• Announcements are true

• Agents can imagine pending messages

If one of these assumptions is dropped, the circularity problem is easily solved: if announcements do not need to be true, then all states are consistent; if announcements are only propositional formulas, consistency of a state \((w, \sigma, c)\) can be trivially checked by evaluating all propositional formulas in \(\sigma\) in the world \(w\). The last point is only a little bit less obvious: if agents cannot imagine pending announcements, then the definition of the pre-accessibility relation \(R_a\) for agent \(a\) (see Definition \[51\]) is that \((w, \sigma, c)R_a(w', \sigma', c')\) if \(wR_aw'\), \(c(a) = c'(a)\) and \(\sigma|_{c(a)} = \sigma'\): the only sequence of announcements that she considers possible is the one she has already received. In that case, the length of the sequence of announcements \(|\sigma|\) together with the size of the formula to evaluate can be used to define truth conditions for consistency and for formulas by induction. Indeed, evaluating a formula \(K_a\varphi\) in a state \((w, \sigma, c)\) only requires evaluating the consistency of states \((w', \sigma', c')\) such that \(|\sigma'| \leq |\sigma|\), which in turn only requires evaluating formulas \(\psi \in \sigma\) in states \((w', \sigma'', c')\) where \(\sigma''\) is a strict prefix of \(\sigma\).

We also note that it is possible to solve the circularity problem by only constraining the last assumption instead of completely dropping it. Indeed, under a bounded non-Zeno behaviour assumption (only a bounded finite number of discrete events occur in a finite time), and assuming a global clock that is common knowledge, the imagination of the agents is sufficiently constrained to solve the circularity problem rather easily (see \[KMS17\]).

In relation with the above discussion, we point out that the circularity problem does not depend on the following assumptions:

• Announcements are broadcast

• Announcements are made by an external source

• Announcements are received in FIFO order.

In Section \[9.3\] we will describe several restricted settings in which we manage to overcome this problem. But first we present a small example to better understand the intuitions behind our logic.

### 9.2.5 Example

We consider two agents \(AGT = \{B, C\}\), where \(B\) stands for Bonnie and \(C\) for Clyde. Bonnie and Clyde go to rob a bank, and Bonnie stays in the car while Clyde goes to the vault. At noon, Bonnie notices that Clyde left the paper with the secret code to open the vault in the car. She uses her smartphone to broadcast the code on their chat group. But Clyde has also realized that he forgot the paper, and before he receives Bonnie’s message he sends a message saying that he does not know the code.

In the following, let \(p\) represent the fact that the secret code is 0000, and \(q\) the fact that the vault is open. The situation at noon is represented by the following initial pointed Kripke model \((M, w)\):

\[
\begin{array}{cccc}
\begin{array}{c}
\text{u: p, q} \\
\hline \hline
B, C
\end{array} & \text{B} & \begin{array}{c}
\text{w: p, \neg q} \\
\hline \hline
B, C
\end{array} & \text{C} & \begin{array}{c}
\text{v: \neg p, \neg q} \\
\hline \hline
B, C
\end{array}
\end{array}
\]

In the following, let \(p\) represent the fact that the secret code is 0000, and \(q\) the fact that the vault is open. The situation at noon is represented by the following initial pointed Kripke model \((M, w)\):
In the initial world \(w\), Bonnie knows \(p\), i.e., she knows the code, but Clyde does not. On the other hand, Clyde knows that \(q\) is not true, i.e., he knows that the vault is closed, but Bonnie does not (Clyde could have memorized the code before leaving the car, and thus he might have opened the vault). In fact, we observe that initially Bonnie knows that “the vault is open if, and only if, Clyde knows the code” or, in epistemic logic, \(K_B(q \leftrightarrow KC_p)\).

The initial state at noon is \(w^B_{c}\). In our scenario, first \(p\) is announced by Bonnie, and then \(\neg KC_p\) is announced by Clyde. The actual state becomes \(w^B_{c}\). Intuitively, since only true announcements are made, we see that \(\neg KC_p\) can only be announced before Clyde receives the announcement of \(p\). We would like to verify whether, after Bonnie receives both announcements but Clyde receives neither (the signal inside the bank is weak), that is in state \(w^B_{c}\), Bonnie knows \(\neg q\), i.e., does she know that the vault is not open, and what does she know about Clyde’s knowledge. In fact, we can prove that the following holds:

\[
(w, \epsilon, 0) \models (p)_{\text{async}}(\neg KC_p)_{\text{async}} \bigcirc B \bigcirc B(K_B \neg q \land \neg K_B KC_p \land \neg K_B \neg KC_p) \quad (9.4)
\]

The meaning is that, after both announcements have been made and received by Bonnie but not by Clyde,

- Bonnie knows that the vault is not open: intuitively, because Clyde told her he didn’t have the code, and thus could not have opened it (recall that \(q\) represents the situation at noon, i.e., before Bonnie announced \(p\)). In state \(w^B_{c}\), all consistent possible states for \(B\) are of the form \((w, \sigma, c)\) for some \(\sigma\) and \(c\): they share the same world \(w\) in which \(q\) does not hold. Indeed, Bonnie initially considers world \(u\) possible, but states with world \(u\) and announcement \(\neg KC_p\) are not consistent. Therefore, Bonnie knows \(\neg q\).

- Bonnie does not know whether Clyde knows the code (because she does not know whether Clyde received her message).

In state \(w^B_{c}\), Bonnie considers state \(w^B_{c}\) as possible, and in this state Clyde knows \(p\). But Bonnie also considers possible state \(w^B_{c}\), in which Clyde considers state \(w^B_{c}\) possible, and thus does not know \(p\).

We note that this example highlights the differences between asynchronous broadcast logic and (synchronous) public announcement logic. Since sending and receiving occur at the same time in PAL, we can informally translate the asynchronous broadcast formula

\[
(p)_{\text{async}}(\neg KC_p)_{\text{async}} \bigcirc B \bigcirc B(K_B \neg q \land \neg K_B KC_p \land \neg K_B \neg KC_p)
\]

to the PAL formula

\[
(p)(\neg KC_p)(K_B \neg q \land \neg K_B KC_p \land \neg K_B \neg KC_p).
\]

Assuming the same initial Kripke model and state \(w\), we first notice that in PAL any formula of the form \((p)(\neg KC_p)\) is false because after a proposition \(p\) is announced, \(KC_p\) holds in any circumstances, so that \(\neg KC_p\) cannot be announced. We can try to simulate the state
where Bonnie has received both announcements but Clyde received neither by using private announcements, made to Bonnie but not to Clyde. Consider the trivial translation of our \( L_{async} \) formula into a formula with private announcements:

\[
(p)_{private_B} (K_C p)_{private_B} (K_B \neg q \wedge \neg K_B K_C p \wedge \neg K_B \neg K_C p),
\]

where \((\varphi)_{private_B}\) means that \(\varphi\) is sent to \(B\) and not to \(C\). In all versions of PAL with any variant of synchronous private or semi-private announcement (e.g. [GG97, BMS98b, BM04, BDM08]), this formula is still false in \((\mathcal{M},w)\): because Bonnie receives \(\neg K_C p\) immediately when it is sent, Clyde cannot receive \(p\) between the announcement of \(\neg K_C p\) and its reception by Bonnie, so that Bonnie knows that Clyde does not know \(p\). Thus, this example shows that there is no obvious translation from asynchronous broadcast logic to a variant of DEL, and that asynchronous broadcast logic is indeed quite different from DEL.

The proof of (9.4) can be found in [KMS17]. Note that here we anticipate the fact that we are in one of the cases where we can solve the circularity problem: indeed, all announcements are in the existential fragment of our language (see Section 9.3.2).

### 9.3 Solving the circularity problem

In this section we show how we solve the circularity problem identified in the last section for several restricted cases.

#### 9.3.1 When the initial model is a finite tree

If we assume in the initial model \(\mathcal{M} = (W, \{R_a\}_{a \in AGT}, w)\) the relation \(\bigcup_a \rightarrow_a\) forms a finite tree over \(W\), then the circularity problem can be avoided. The reader should be aware that such an initial model is nonsensical: in terminal worlds, \(K_a \bot\) holds for all agents \(a\). In this case, we can define a well-founded order on tuples of the form \((w, \sigma, c, \varphi)\), where \(\varphi\) is either a formula in \(L_{async}\) or \(\checkmark\), the idea being that a tuple \((w, \sigma, c, \varphi)\) means ‘\(w, \sigma, c \models \varphi\)’.

**Definition 53** The order \(\prec\) is defined as follows:

1. \((w, \sigma, c, \varphi) \prec (w', \sigma', c', \varphi')\) if either
   1. \(w\) is a descendent of \(w'\) in \(\mathcal{M}\),
   2. \(w = w'\) and \(|\sigma| + |\varphi| < |\sigma'| + |\varphi'|\),
   3. \(w = w'\), \(|\sigma| + |\varphi| = |\sigma'| + |\varphi'|\) and \(c < c'\),

where \(|\checkmark| = 1\).

It is clear that \(\prec\) is a well-founded order, and with this order Figure 9.2 forms a well-founded inductive definition of consistency and semantics of our language.

We detail the non-trivial cases. For the second clause of Figure 9.2 observe that by Point 3 of Definition 53 if \(c' < c\) then \((w, \sigma, c', \checkmark) \prec (w, \sigma, c, \checkmark)\), and for all \(w, \sigma', c\) and \(\psi\), by Point 2 of Definition 53 we have \((w, \sigma', c, \psi) \prec (w, \sigma'::\psi, c, \checkmark)\).

For the clause for \(K_a \varphi\) of Figure 9.2 by Point 1 of Definition 53 we have that for all \(\varphi, \sigma, \sigma', c, c'\), if \(w'\) is a child of \(w\) then \((w', \sigma', c', \varphi) \prec (w, \sigma, c, K_i \varphi)\).

Finally, for the clause for \(\langle \psi \rangle_{async} \varphi\) of Figure 9.2 by Point 2 of Definition 53 we have that \((w, \sigma::\psi, c, \varphi) \prec (w, \sigma, c, \langle \psi \rangle_{async} \varphi)\) for all \(w, \sigma, c, \varphi\) and \(\psi\) (note that \(|\langle \psi \rangle_{async} \varphi| = 1 + |\psi| + |\varphi|\)).

**Example 47** Suppose that we have only one agent \(a\). Let us consider initial model \(\mathcal{M}\):
In this model, \( p \) holds in the actual world \( w \), but agent \( a \) does not know it. Assume that \( p \) can be announced at least once (\( p \in \mathcal{A} \)). We show that, as expected, after \( p \) is announced and agent \( a \) receives this announcement, agent \( a \) knows that \( p \) holds. Formally, we prove that, in \( \mathcal{M} \otimes \mathcal{A} \), we have \((w, \epsilon, 0) \models (p)_{\text{async}} \Diamond_a K_a p\). To do so we in fact show that \((w, [p], a \rightarrow 1) \models K_a p\), from which it follows that \((w, [p], 0) \models \Box_a K_a p\), hence the desired result. By Definition 51 for pre-accessibility relations, every state \( S \) such that \((w, [p], a \rightarrow 1)R_a S\) is of the form \( S = (w', p; \sigma, a \rightarrow 1)\), where \( w' \in \{u, v\} \) and \( \sigma \) is a sequence of announcements. We must show that every such state either is inconsistent or satisfies \( p \).

First, for \( w' = u \). According to the clause for \( p \) in Figure 9.2, we have that \((u, \epsilon, 0) \not\models p\), and by the second clause in Figure 9.3 it follows that \((u, [p], 0) \not\models \checkmark\), from which it also follows also that \((u, [p], a \rightarrow 1) \not\models \checkmark\) and \((u, p; \sigma, a \rightarrow 1) \not\models \checkmark\), for any \( \sigma \).

Now, for \( w' = v \), by the first clause for \( p \) in Figure 9.3, it follows that for all states of the form \( S = (v, p; \sigma, a \rightarrow 1)\), \( S \models p\), so that finally every state related to \((w, [p], a \rightarrow 1)\) is either inconsistent or verifies \( p \). Note that we could also prove that \( S \) is consistent.

In practice, this setting can be used as an approximation scheme: taking the tree unfolding of models and cutting them at level \( \ell \) amounts to assuming that agents cannot reason about deeper nesting of knowledge. This approach is similar to the well known idea of bounded rationality [Jon99], where it is assumed that due to computational limits, agents have only approximate, bounded information about other agents’ knowledge, which is represented by allowing only finite-length paths in the Kripke model. We point out, however, that this method of approximation is only appropriate in certain settings. One issue is that it does not allow the accurate representation of transitive accessibility relations, where the leaves of an initial model of any depth \( \ell \) may be reached just by evaluating a formula with one knowledge operator. This setting calls for more work to clarify what the finite tree restriction really captures.

### 9.3.2 Announcing existential formulas

Now, we again allow the initial model to be arbitrary. In particular, we may use one of the common models of knowledge, for example an initial model whose underlying frame is KD45 (relations are serial, transitive and Euclidean) or S5 (relations are equivalence relations); see [EMHV03a]. However, we restrict the announcement protocol to the existential fragment of our logic, generated by the following rule:

\[
\varphi ::= p \mid \neg p \mid (\varphi \land \varphi) \mid (\varphi \lor \varphi) \mid K_a \varphi \mid \Box_a \varphi \mid (\varphi)_{\text{async}} \varphi
\]

where \( p \) ranges over \( \mathcal{A} \) and \( a \) ranges over \( \mathcal{AGT} \). Formulas of the existential fragment are called **existential formulas**. If an announcement protocol contains only existential formulas, we call it an **existential announcement protocol**. For instance, the announcement protocol in Example 42 is existential.

Here we tackle the circularity problem by defining consistency and truth conditions separately. We first define as a fixed point the semantics of existential announcements in \( \mathcal{A} \), together with consistency. In a second step we define the semantics of the full logic with existential announcements as described in Figure 9.2 using the fixed point to evaluate consistency.

We fix an initial model \( \mathcal{M} = (W, \{R_a\}_{a \in \mathcal{AGT}}, w) \) and an existential announcement protocol \( \mathcal{A} \). Let \( B \) be the set of all pairs \((S, \varphi)\) such that \( S \) is a state of \( \mathcal{M} \otimes \mathcal{A} \) and \( \varphi \) is either a formula in \( \mathcal{A} \) or \( \checkmark \), the symbol for consistency. Observe that \((P(B), \subseteq)\) forms a complete lattice. We
now consider the function \( f : \mathcal{P}(B) \rightarrow \mathcal{P}(B) \) defined in Figure 9.3. Function \( f \) takes a set \( \Gamma \) of truth pairs (pairs \((S, \varphi)\) such that \( S \models \varphi \)), and extends it with the new truth pairs that can be inferred from \( \Gamma \) by applying each of the rules in Figure 9.2 once. For instance, if \((w, \sigma, c, \varphi) \in \Gamma\) and \((w, \sigma, c, \psi) \in \Gamma\), then \((w, \sigma, c, \varphi \land \psi) \in \Gamma\). Similarly, if \((w, \sigma, c, \varphi) \in \Gamma\) or \((w, \sigma, c, \psi) \in \Gamma\), then \((w, \sigma, c, \varphi \lor \psi) \in \Gamma\). It would thus no longer hold that \( f(\Gamma_1) \subseteq f(\Gamma_2) \) whenever \( \Gamma_1 \subseteq \Gamma_2 \). As \( f \) is clearly a decreasing function, we would not be able to apply the Knaster-Tarski theorem.

Remark 4 The Knaster-Tarski theorem is often used to define the denotational semantics of programming languages [Win93] in the same spirit as what we do here to define consistency.

9.4 Semantic properties

In this section we establish some semantic properties of our logic. First we compare it with PAL, explaining why \( L_{async} \) is not a conservative extension of PAL when we have at least two

\[ f(\Gamma) = \Gamma \cup \{(w, \sigma, c, p) \mid p \in \mathcal{U}(w) \} \]

\[ \cup \{(w, \sigma, c, \neg p) \mid p \notin \mathcal{U}(w) \} \]

\[ \cup \{(w, \sigma, c, (\varphi \land \psi)) \mid (w, \sigma, c, \varphi) \in \Gamma \text{ and } (w, \sigma, c, \psi) \in \Gamma \} \]

\[ \cup \{(w, \sigma, c, (\varphi \lor \psi)) \mid (w, \sigma, c, \varphi) \in \Gamma \text{ or } (w, \sigma, c, \psi) \in \Gamma \} \]

\[ \cup \{(w, \sigma, c, \neg \varphi) \mid (w, \sigma, c, \varphi) \in \Gamma \} \]

\[ \cup \{(w, \sigma, c, \land \varphi) \mid c(i) < |\sigma| \text{ and } (w, \sigma, c^{+a}, \varphi) \in \Gamma \} \]

\[ \cup \{(w, \sigma, c, \land \psi) \mid \sigma::\psi \in \text{Seq}(A), (w, \sigma, c, \psi) \in \Gamma \text{ and } (w, \sigma::\psi, c, \varphi) \in \Gamma \} \]
agents. Then we establish some validities of $L\text{async}$ that show how the correctly defined semantics captures the intuitions we have about asynchrony.

For the rest of the section, we assume that we have a class of initial models and a class of announcement protocols for which the circularity problem can be solved and the semantics defined as in Figure 9.2 (for example, arbitrary initial models and positive announcements), and we discuss some validities of our logic.

### 9.4.1 Difference from PAL

We discuss the difference between the semantics of our logic and those of public announcement logic (PAL). In PAL, every time an announcement is made, the Kripke model is updated by removing possible worlds where the announcement is not true. This amounts to using the new information to delete epistemic alternatives that are no longer considered possible: since announcements are true, a world where an announcement is not true is not a possible world. In our case, epistemic alternatives cannot be deleted at the time of the announcement, since announcements are not received immediately by the agents, and in general agents can even have an unbounded number of pending announcements to read. Instead, this pruning is performed directly in the semantics of the knowledge operator, by eliminating all possible states that are not consistent: the pruning is not performed at the moment of the announcement, but is delayed until a knowledge operator is evaluated. Thus, the update operation in PAL and the consistency check in our logic play the same role. This is also reflected in the circularity problem, which stems from a mutual dependence between the definition of the semantics and, in the case of PAL, that of the update, and in the case of our logic, that of consistency.

We also note that if there are at least two agents, our logic is not a conservative extension of PAL. An intuitive way of seeing this is that in PAL, an announcement immediately becomes common knowledge, which we recall, means that all agents know $p$, they all know that they all know $p$, they all know that they all know that they all know $p$, and so on. On the other hand, in asynchronous systems common knowledge cannot be reached [HM90a, MT88], and our logic illustrates this phenomenon. Even finite approximations of common knowledge fail to hold in our logic. For instance, while $[p]_{\text{PAL}}K_aK_bK_pp$ is a validity of PAL, its translation $[p]_{\text{async}}\Box_a\Box_bK_aK_bK_pp$ is not valid.

**Proposition 14** There exist $M$, an announcement protocol $A$ and a consistent state $S \in M \otimes A$ such that $M \otimes A, S \not\models [p]_{\text{async}}\Box_a\Box_bK_aK_bK_pp$.

**Proof.**

The idea is the following: after announcing $p$, and after all agents have received the message $p$, $a$ does not know whether agent $b$ has received $p$. Therefore, $a$ does not know that $b$ knows $p$. Let us consider the following initial model $M$: ...
The actual world is $w$. Since $p$ holds in $w$, it can be announced. Let $A = \{[p]\}$. We prove that $M \otimes A, (w, \epsilon, 0) \not\models [p] \bigcirc_a \bigcirc_b K_a K_b p$. To see this, observe that after $p$ has been announced and received by agent $a$ and agent $b$ (i.e., after evaluation of the first three operators of the formula), we reach state $S = ([w], \frac{a}{b} \Rightarrow \frac{1}{1})$. But in $M \otimes A$, we have (we only represent a part of $M \otimes A$, which is infinite):

Indeed in state $S = ([w], \frac{a}{b} \Rightarrow \frac{1}{1})$ agent $a$ considers it possible that agent $b$ did not receive announcement $p$, and thus she considers state $S' = ([w], \frac{a}{b} \Rightarrow \frac{1}{0})$ possible. In $S'$, because $b$ received no announcement, and in the initial model we have $w R_a [b] u$, agent $b$ considers it possible that the actual world is $u$ and nothing has been announced, i.e., she considers state $S'' = ([u], \epsilon, 0)$ possible. Because $p$ does not hold in $S''$, we have that $S \not\models K_a K_b p$, which concludes the proof.

### 9.4.2 Validities

We say that a formula $\varphi$ is valid if for every initial model $M$ and every announcement protocol $A$ in the classes considered, and for every consistent state $S \in M \otimes A$, we have $M \otimes A, S \models \varphi$. We write $\models \varphi$ to express that $\varphi$ is valid. In the following proposition we establish some validities that provide insights into our framework and show how our definitions correctly capture some natural properties that intuitively should hold in the asynchronous framework we consider.

**Proposition 15** For every $\varphi \in L_{async}$ and propositional formula $\psi$, we have that:

1. $\models K_a \bigcirc_b \varphi \leftrightarrow \bigcirc_b \bigcirc_a \varphi$
2. $\models K_a T \rightarrow (\bigcirc_a \varphi \leftrightarrow \neg \bigcirc_a \neg \varphi)$
3. $\models \neg \bigcirc_a T \rightarrow [\psi] \bigcirc_a K_a \psi$
4. $\models \neg \bigcirc_a T \rightarrow [\psi] \bigcirc_a (\neg \bigcirc_a T \rightarrow K_a \psi)$

**Proof.**

We prove the first validity and the other three are left to the reader.

Suppose that we have $M \otimes A, (w, \sigma, c) \models \bigcirc_a \bigcirc_b \varphi$. By Figure 9.2, this means that $c(1) < |\sigma|$ and $M \otimes A, (w, \sigma, c^{+a}) \models \bigcirc_b \varphi$, and the latter implies that $c^{+b}(b) < |\sigma|$ and $M \otimes A, (w, \sigma, (c^{+a})^{+b}) \models \varphi$. Now, because $(c^{+a})^{+b} = (c^{+b})^{+a}$, we obtain that $M \otimes A, (w, \sigma, (c^{+b})^{+a}) \models \bigcirc_a \varphi$, and therefore $M \otimes A, (w, \sigma, c) \models \bigcirc_b \bigcirc_a \varphi$. The proof for the other direction is symmetric. 

Let us comment on these validities. The first one says that it is possible to permute the order of agents that receive next announcements in their respective queues. The second one says that if an agent has an announcement to read, then reading it is a deterministic operation. The third one says that if an agent has no pending announcement and some propositional formula is announced, then after reading his next pending announcement, the agent will know that formula.
Let \( \psi \) be the content of the agent’s channel, so that the possibility that \( \phi \) is not possible as \( a \) is under the scope of a knowledge operator, then its truth value is left unchanged by the announcement of any formula \( \psi \). Indeed, the knowledge operator considers all possibilities for the content of the agent’s channel, so that the possibility that \( \psi \) is in the channel is considered, whether \( \psi \) was actually announced or not.

In the following, in addition to the assumption that models and announcement protocols are restricted to classes for which the semantics is defined, we consider announcement protocols in which each announcement can be made infinitely many times. We call such protocols free protocols.

**Proposition 16** Let \( \phi \) be a formula in \( L_{async} \), in which every \( \Box_a \) is under the scope of some \( K_b \), and let \( A_\infty \) be a free protocol. For every initial model \( M \) and consistent state \( S = (w, \sigma, c) \in M \otimes A_\infty \), for every \( \psi \in A_\infty \), we have \( M \otimes A_\infty, S \models \langle \psi \rangle_{async} \psi \wedge \phi \).

This result follows immediately from the following lemma:

**Lemma 5** Let \( \phi \) be a formula in \( L_{async} \), in which every \( \Box_a \) is under the scope of some \( K_b \), and let \( A_\infty \) be a free protocol. For every initial model \( M \) and for every consistent state \( (w, \sigma, c) \in M \otimes A_\infty \), for every \( \psi \in A_\infty \) such that \( (w, \sigma; \psi, c) \) is consistent, we have \( M \otimes A_\infty, (w, \sigma; \psi, c) \models \phi \) iff \( M \otimes A_\infty, (w, \sigma, c) \models \phi \).

**Proof.**

By induction on \( \phi \). The Boolean cases are trivial.

Case \( \psi = K_w \phi' \): Since \( (w, \sigma, c) \) is a state, \( c(a) \leq |\sigma| \). It is then easy to check that \( \{S \mid (w, \sigma; \psi, c) R_a S\} = \{S \mid (w, \sigma, c) R_a S\} \), and the result follows.

Case \( \psi = \langle \psi \rangle_{async} \phi' \): If \( \psi' \notin A_\infty \), the formula \( \phi \) trivially does not hold in both states. Otherwise, because \( \psi' \) has infinite multiplicity in \( A_\infty \), \( \sigma; \psi' \in \text{Seq}(A_\infty) \). We therefore have \( (w, \sigma; \psi, c) \models \langle \psi \rangle_{async} \phi' \) iff \( (w, \sigma; \psi, c) \models \psi' \) and \( (w, \sigma; \psi', c) \models \phi' \).

Assume that \( (w, \sigma; \psi, c) \models \langle \psi \rangle_{async} \phi' \), we prove that \( (w, \sigma, c) \models \phi \) and \( (w, \sigma; \psi', c) \models \phi' \). We have that \( (w, \sigma; \psi, c) \models \psi' \) (hence \( (w, \sigma; \psi; \psi', c) \) is consistent) and \( (w, \sigma; \psi; \psi', c) \models \phi' \). Because \( \psi' \) is a subformula of \( \psi \), each \( \Box_a \) in it is in the scope of some \( K_b \); we can thus apply the induction hypothesis for \( (w, \sigma; \psi, c) \models \psi' \), obtaining that \( (w, \sigma, c) \models \psi' \). By induction hypothesis on \( (w, \sigma; \psi; \psi', c) \models \phi' \), we get first that \( (w, \sigma; \psi, c) \models \phi' \), then \( (w, \sigma, c) \models \phi' \) and finally \( (w, \sigma; \psi', c) \models \phi' \) (observe that \( (w, \sigma; \psi', c) \) is consistent since \( (w, \sigma, c) \models \psi' \)). We have proved that \( (w, \sigma, c) \models \langle \psi' \rangle_{async} \phi' \).

The other direction is treated the same way.

Finally, the case \( \phi = \Box_a \phi' \) is not possible as \( \Box_a \) is not under the scope of any \( K_b \).

### 9.5 Model checking

Here we address the model checking problem when \( A \) is a finite multiset, that is, when the support set of \( A \) is finite and the multiplicity of each element is an integer. More precisely, we consider the following decision problem:
input: an initial pointed model \((\mathcal{M}, w)\), a finite multiset of formulas \(\mathcal{A}\) (where multiplicities are written in unary), a formula \(\varphi_0\);

output: yes if \(\mathcal{M} \otimes \mathcal{A}, (w, \epsilon, 0) \models \varphi_0\), no otherwise.

In practice, model checking is used to check a scenario described by \(\mathcal{A}\) and \(\varphi_0\) from a given initial situation \((\mathcal{M}, w)\).

### 9.5.1 Propositional announcements

In this section, we suppose that formulas in \(\mathcal{A}\) are propositional. Note that in this case (which is a particular case of existential announcements) the circularity problem does not exist, as consistency of a state \((w, \sigma, c)\) can be trivially checked by verifying that all propositional formulas in \(\sigma\) hold in world \(w\) of the initial model, according to the classic semantics of propositional logic.

We consider the model checking problem for \(L_{async}\) where inputs \((\mathcal{M}, w, \mathcal{A}, \varphi_0)\) are such that \(\mathcal{A}\) only contains propositional formulas. We call this problem the model checking problem for propositional protocols.

**Theorem 17** The model checking problem for propositional protocols is in Pspace.

**Proof.**

Figure 9.4 presents an algorithm that takes a pointed model \((\mathcal{M}, w)\), a finite multiset \(\mathcal{A}\), a sequence \(\sigma \in \text{Seq}(\mathcal{A})\), a cut \(c\) on \(\sigma\) and a formula \(\varphi\) as an input. To check the consistency of a state \((w, \sigma, c)\), we call \(\text{checkconsistency}(\mathcal{M}, \mathcal{A}, w, \sigma, c)\) which verifies that every (propositional) formula \(\psi\) occurring in \(\sigma\) evaluates to true with the valuation \(w(w)\). We leave it to the reader to write the pseudo-code of the function \(\text{checkconsistency}\).

It is easily proven by induction that, for all \(\psi\), the following property \(P(\varphi)\) holds:

\[
\mathcal{M}, \mathcal{A}, (w, \sigma, c) \models \varphi \iff \text{mc}(\mathcal{M}, \mathcal{A}, w, \sigma, c, \varphi) \text{ returns true.}
\]

This establishes the correctness of the algorithm. We now analyze its complexity.

First, observe that because \(\mathcal{A}\) is finite and each element has finite multiplicity, we have that \(\text{Seq}(\mathcal{A})\) only contains sequences of length linear in \(|\mathcal{A}|\) (recall that multiplicities are written in unary). It is therefore easy to see that the consistency check \((\ast \checkmark)\) is done in polynomial time in the size of the input and thus requires a polynomial amount of space. Now, the number of nested calls of \(\text{mc}\) is bounded by the size of the formula to check, and each call requires a polynomial amount of memory for storing local variables, so that the algorithm runs in polynomial space. ■

**Theorem 18** The model-checking problem for propositional protocols is PSPACE-hard.

**Proof.**

See [KMS17]. ■

### 9.5.2 Finite tree initial model

In this section, we restrict the set of inputs \(\mathcal{M}, \mathcal{A}, w, \varphi_0\) of the model checking problem to those where the initial pointed models \((\mathcal{M}, w)\) are finite trees rooted in \(w\).

**Theorem 19** The model-checking problem when we restrict initial models to finite trees is in Pspace.
function \text{mc}(\mathcal{M}, A, w, \sigma, c, \varphi)

match \varphi do
    case p: return \ p \in V(w);
    case \checkmark: return \text{checkconsistency}(\mathcal{M}, A, w, \sigma, c)
    case \neg\psi: return \neg \text{mc}(\mathcal{M}, A, w, \sigma, c, \psi);
    case (\psi_1 \land \psi_2): return \text{mc}(\mathcal{M}, A, w, \sigma, c, \psi_1) \land \text{mc}(\mathcal{M}, A, w, \sigma, c, \psi_2);
    case \text{K}_a \psi:
        for (u, \sigma', c') such that \ wR_a u, \sigma' \in \text{Seq}(A) and \ c' \ is \ a \ cut \ on \ \sigma' \ do
            if \ \sigma'[1..c(a)] = \sigma[1..c'(a)] \ and \ \text{mc}(\mathcal{M}, A, u, \sigma', c', \checkmark) \ then
                return false
            endIf
        endFor
        return true
    case \langle \psi \rangle \chi:
        if \sigma::\psi \in \text{Seq}(A) \ and \ \text{mc}(\mathcal{M}, A, w, \sigma, c, \psi) \ then
            return \text{mc}(\mathcal{M}, A, w, \sigma::\psi, c, \chi);
        else
            return false;
        endIf
    case \text{O}_a \psi: return c(a) < |\sigma| \ and \ \text{mc}(\mathcal{M}, A, w, \sigma, c^+a, \psi)
endFunction

Figure 9.4: Model checking algorithm

Proof.

We consider the algorithm of Figure 9.4 again but now the function \((\ast \checkmark)\) is different than the one used in the proof of Theorem 17. The consistency checking \((\ast \checkmark)\) consists of calling the following function:

function \text{checkconsistency}(\mathcal{M}, A, w, \sigma, c)

if \ c = 0
    return true
else
    for \ c' < c \ do
        if \ \text{mc}(\mathcal{M}, A, w, \sigma, c', \checkmark) \ then
            return true
        endIf
    endFor
    return \text{mc}(\mathcal{M}, A, w, \sigma', c, \psi) \ where \ \sigma = \sigma'::\psi
endFunction

Soundness and completeness are proven by induction on inputs using the order \(\prec\) defined in Section 9.3.1.

Concerning the complexity, the argument given in the proof of Theorem 17 no longer holds. In order to bound the number of nested calls of \text{mc}, we have to remark that from a call of \text{mc} to a sub-call of \text{mc}:

1. either we change the current world \(w\) in the initial model for a successor \(u\) in the finite tree;

2. or the quantity \(|\sigma| + |\varphi| + \sum_{a \in \text{AGT}} c(a)\) is strictly decreasing, where \(|\varphi|\) is the length of \(\varphi\).
and if \( \sigma = [\varphi_1, \ldots, \varphi_k] \) then \( |\sigma| = \sum_{i=1}^{k} |\varphi_i| \).

Now, the number of times (1) occurs is bounded by the depth \( \text{depth}(M, w) \) of the finite tree \( M, w \). As each \( \varphi \) is either a subformula of the input formula \( \varphi_0 \) or a subformula of a formula in \( A, |\varphi| \leq |\varphi_0| + |A| \) where \( |A| := \sum_{\psi \in A} |\psi| \), and where each single formula \( \psi \) is counted as many times as it occurs in the multiset \( A \). Furthermore, \( |\sigma| \leq |A| \) and \( c(a) \leq |A| \). Thus, the quantity \( |\sigma| + |\varphi| + \sum_{\varphi \in \text{AGT}} c(a) \) is bounded by \( (|\text{AGT}| + 2)|A| + |\varphi_0| \). Therefore, the number of nested calls to \( \text{mc} \) is bounded by \( \text{depth}(M, w) \times ((|\text{AGT}| + 2)|A| + |\varphi_0|) \). So the algorithm requires a polynomial amount of memory in the size of the input (recall that the multiplicity of \( A \) is encoded in unary). ■

### 9.5.3 Existential announcements

In this subsection, we design an exponential-time algorithm for the model checking problem in the case of existential announcements.

Given an input \( M, A, w, \varphi_0 \), the algorithm first computes the least fixed point \( \Gamma^* \) of the function \( f \) defined in Section 9.3.2. Because the number of possible sequences in \( \text{Seq}(A) \) is exponential in \( |A| \), the set \( B \) of pairs \((\mathcal{S}, \varphi)\) where \( \mathcal{S} \in M \otimes A \) and \( \varphi \in A \cup \{\checkmark\} \) is of size exponential in the size of the input, and therefore computing the fixed point requires exponential time in the size of the input. This gives us the semantics of consistency for states of \( M \otimes A \).

Then, to evaluate \( \varphi_0 \), we use the procedure \( \text{mc} \) of Figure 9.4 where line (\( \ast \checkmark \)), which checks the consistency of a state \((w, \sigma, c)\), is replaced by checking whether \((w, \sigma, c, \checkmark) \in \Gamma^* \). The algorithm \( \text{mc} \) also requires exponential time. To sum up:

**Theorem 20** The model-checking problem for existential announcements is in \( \text{Exptime} \).

### 9.6 Satisfiability for propositional announcements

In this section, we address the satisfiability problem when \( A \) is a finite multiset of propositional formulas, that is, when the support set of \( A \) is finite, the multiplicity of each element is an integer and formulas in \( A \) are propositional. More precisely, we say that a formula \( \varphi_0 \) is \( A \)-satisfiable if there exists an initial pointed model \((M, w)\) such that \( M \otimes A, (w, \epsilon, 0) \models \varphi_0 \). We consider the following decision problem:

- **input:** a finite multiset of propositional formulas \( A \) (where multiplicities are written in unary), a formula \( \varphi_0 \);
- **output:** yes if \( \varphi_0 \) is \( A \)-satisfiable, no otherwise.

In practice, a typical application of the satisfiability problem would be to check that a class of systems described by a formula \( \varphi \) satisfies a property \( \psi \). To do so, one checks whether \( \varphi \land \neg \psi \) is satisfiable. If it is not, then indeed all \( \varphi \)-systems satisfy \( \psi \). If it is satisfiable, then the algorithm we present here (like all tableau methods) produces a counter-example, i.e., a model \((M, w)\) such that \( M \otimes A, (w, \epsilon, 0) \not\models \varphi \land \neg \psi \), or in other words, a \( \varphi \)-system that does not satisfy \( \psi \).

#### 9.6.1 Tableau method description

Our tableau method manipulates terms that we call **tableau terms**, which are of the following kind:
• \((w \sigma c \varphi)\): \(w\) is a world symbol that represents a world of the model \(M\) being constructed, \(\sigma\) is a sequence of formulas in Seq\((A)\), \(c\) is a cut for \(\sigma\) and \(\varphi\) is a sub-formula of \(\varphi_0\) that should be true in \(M \otimes A, (w, \sigma, c)\).

• \((wR_a u)\): \(w\) and \(u\) are two world symbols such that \(wR_a u\) in the model \(M\) being constructed.

• \(\bot\): Denotes an inconsistency.

A tableau rule is represented by a numerator \(N\) above a line and a finite list of denominators \(D_1, \ldots, D_k\) below this line, separated by vertical bars, representing nondeterministic choice:

\[
\frac{N}{D_1 | \ldots | D_k}
\]

The numerator and the denominators are finite sets of tableau terms.

A tableau for input \((A, \varphi_0)\) is a finite tree with a set of tableau terms at each node, whose root is:

\[
\Gamma_0 = \{(w_0, \epsilon, 0, \varphi_0)\}
\]

A rule with numerator \(N\) is applicable to a node carrying a set \(\Gamma\) if \(\Gamma\) contains an instance of \(N\) for which the rule has not yet been applied. If no rule is applicable, \(\Gamma\) is said to be saturated. We call a node \(n\) an end node if the set of tableau terms \(\Gamma\) it carries is saturated, or if \(\bot \in \Gamma\). The tableau is extended the following way:

1. Choose a leaf node \(n\) carrying \(\Gamma\) where \(n\) is not an end node, and choose a rule applicable to \(n\).

2. For each denominator \(D_i\) of the rule, create one successor node for \(n\) carrying the union of \(\Gamma\) with an appropriate instanciation of \(D_i\).

A branch in a tableau is a path from the root to an end node. A branch is closed if its end node contains \(\bot\), otherwise it is open. A tableau is closed if all its branches are closed, otherwise it is open. A pair \((A, \varphi_0)\) is said to be consistent if no tableau for \((A, \varphi_0)\) is closed.

The tableau rules are described in Figure 9.5, in which we write \((\sigma, c) \sim_a (\sigma', c')\) for \(\sigma|_{c(a)} = \sigma'|_{c'(a)}\).

Remark 5 Rule ch for choosing valuations is necessary for checking consistency of states in rules \(K_\varphi \varphi\) and \(\checkmark\). For this reason, rule ch is always applied in priority before rules \(K_\varphi \varphi\) and \(\checkmark\).

In a node carrying \(\Gamma\) and saturated for rule ch, if \(w\) is a world symbol in \(\Gamma\), we say that \(\sigma\) is true in \(w\) if the valuation \(\nu\), defined by \(\nu(p) = 1\) if \((w, \epsilon, 0, p) \in \Gamma\) and \(\nu(p) = 0\) if \((w, \epsilon, 0, \neg p) \in \Gamma\), satisfies every formula in \(\sigma\) (recall that in this section announcements are propositional).

9.6.2 Tableau method soundness and completeness

In this section we prove that the tableau method is sound and complete. Note that we will establish that every tableau is finite in the complexity analysis of the tableau method (see Theorem 21).

Proposition 17 If \((A, \varphi_0)\) is consistent, then \(\varphi_0\) is \(A\)-satisfiable.
### 9.6. Satisfiability for Propositional Announcements

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (w \sigma c \varphi) )</td>
<td>for all atomic propositions ( p ) appearing in ( \varphi_0 ) and ( A )</td>
</tr>
<tr>
<td>( (w \in 0 p) \mid (w \in 0 \neg p) )</td>
<td>( ch )</td>
</tr>
<tr>
<td>( (w \sigma c (\varphi \land \psi)) \land (w \sigma c \varphi) )</td>
<td>(9.6)</td>
</tr>
<tr>
<td>( (w \sigma c \neg(\varphi \land \psi)) \neg \land )</td>
<td></td>
</tr>
<tr>
<td>( (w \sigma c \varphi) )</td>
<td>(9.6)</td>
</tr>
<tr>
<td>( (w \sigma c \neg \varphi) )</td>
<td>(9.6)</td>
</tr>
</tbody>
</table>
| \( (w \sigma c \neg \varphi) \neg \land \) | where \( p \in AP \)
| \( (w \sigma c \bigcirc_a \varphi) \) | if \( c(a) < |\sigma| \) |
| \( (w \sigma c \bigcirc_a \varphi) \bigcirc_a \) | (9.6) |
| \( (w \sigma c \neg \bigcirc_a \varphi) \) | if \( c(a) < |\sigma| \) |
| \( (w \sigma c \neg \bigcirc_a \varphi) \neg \bigcirc_a \) | (9.6) |
| \( (w \sigma c \langle \psi \rangle_{async} \varphi) \) | if \( \sigma \models \psi \in Seq(A) \) |
| \( (w \sigma c \langle \psi \rangle_{async} \) | (9.6) |
| \( (w \sigma c \langle \psi \rangle_{async} \neg \psi) \neg \langle \psi \rangle_{async} \) | if \( \sigma \models \psi \notin Seq(A) \) |
| \( (w \sigma c \langle \psi \rangle_{async} \neg \psi) \) | (9.6) |
| \( (w \sigma c \langle \psi \rangle_{async} \neg \psi) \langle \psi \rangle_{async} \) | if \( \sigma \models \psi \in Seq(A) \) |
| \( (w \sigma c \neg \langle \psi \rangle_{async} \varphi) \) | (9.6) |
| \( (w \sigma c \neg \langle \psi \rangle_{async} \neg \psi) \neg \langle \psi \rangle_{async} \) | if \( \sigma \models \psi \in Seq(A) \) |
| \( (w \sigma c K_a \varphi)(wR_u) \) | \( u \sigma' c' \varphi \) for all \( (\sigma', c') \sim_a (\sigma, c) \) and \( \sigma' \) true in \( u \) (see Remark 4) |
| \( (w \sigma c \neg K_a \varphi) \) | \( u \sigma'_1 c'_1 \varphi(wR_u), \cdots | (u \sigma'_n c'_n \varphi(wR_u) \) |
| \( \neg K_a \varphi \) | (9.6) |
| \( \neg K_a \varphi \) | where \( (\sigma'_i, c'_i) \sim_a (\sigma, c) \) and \( u \) is fresh |
| \( (w \sigma c \varphi) \) | \( \bot \) if \( \sigma \) is not true in \( w \) |
| \( (w \in 0 p)(w \in 0 \neg p) \) | \( \bot \) for \( p \in AP \) |

**Figure 9.5:** Tableau rules
Proof. Suppose that \((A, \varphi_0)\) is consistent, and consider a tableau \(t\) for \((A, \varphi_0)\). By assumption, this tableau is open, which means that it has an open node. Consider one such open branch, and let \(\Gamma\) be the set of tableau terms carried by its end node. We define the model \(M = (W, \{R_a\}_{a \in \text{AGT}}, \psi)\), where

- \(W = \{w \mid (w, \sigma, c, \varphi) \in \Gamma\) for some \(\sigma, c,\) and \(\varphi)\},
- \(R_a = \{(w, u) \mid (w, R_a, u) \in \Gamma\}\) and
- for each \(w \in W\), \(\psi(w) = \{p \mid (w, \epsilon, p) \in \Gamma\}\).

We prove that for all \((w, \sigma, c, \varphi) \in \Gamma\), it holds that \(M \otimes A, (w, \sigma, c) \models \varphi\). Because \((w_0, \epsilon, 0, \varphi_0)\) is in \(\Gamma_0 \subseteq \Gamma\), it follows that \(\varphi_0\) is \(A\)-satisfiable.

If \(\varphi = p\), by saturation of rule \(p\) we have \((w, \epsilon, 0, p) \in \Gamma\), thus \(p \in \psi(w)\) by construction of \(M\), and \(M \otimes A, (w, \sigma, c) \models p\).

If \(\varphi = \neg p\), by saturation of rule \(\neg p\) we have \((w, \epsilon, 0, \neg p) \in \Gamma\). We cannot have \((w, \epsilon, 0, p) \in \Gamma\), otherwise the branch would be closed by saturation of rule \(\bot\). Therefore \(p \notin \psi(w)\), and \(M \otimes A, (w, \sigma, c) \models \neg p\).

For boolean connectives, the result follows by saturation of the appropriate tableau rule, plus application of the induction hypothesis.

If \(\varphi = \bigcirc_a \varphi',\) we have that \(\varphi(a) < |\sigma|\), otherwise \(\Gamma\) would contain \(\bot\) by saturation of rule \(\bigcirc_a\) and the branch would be closed. Therefore, again by saturation of rule \(\bigcirc_a\), \(\Gamma\) contains \((w, \sigma, c^a, \varphi')\). By induction hypothesis we get that \(M \otimes A, (w, \sigma, c^a) \models \varphi',\) and thus \(M \otimes A, (w, \sigma, c) \models \bigcirc_a \varphi'.\)

If \(\varphi = \neg \bigcirc_a \varphi',\) we apply similar reasoning, except for the case \(c(a) = |\sigma|\) in which \(\varphi\) trivially holds.

If \(\varphi = \langle \psi \rangle_{a \text{async}} \varphi',\) then \(\sigma::\psi \in \text{Seq}(A)\), otherwise the branch would be closed. It follows by saturation of rule \(\langle \psi \rangle_{a \text{async}}\) that \((w, \sigma, c, \psi)\) and \((w, \sigma::\psi, c, \varphi')\) are in \(\Gamma\), and we conclude by applying the induction hypothesis.

If \(\varphi = \neg \langle \psi \rangle_{a \text{async}} \varphi',\) in case \(\sigma::\psi\) is not in \(\text{Seq}(A)\), \(\varphi\) trivially holds. Otherwise, by saturation of rule \(\neg \langle \psi \rangle_{a \text{async}}\), either \((w, \sigma, c, \neg \psi)\) is in \(\Gamma\), or both \((w, \sigma, c, \psi)\) and \((w, \sigma::\psi, c, \neg \varphi')\) are in \(\Gamma\).

In both cases, we conclude by induction hypothesis.

If \(\varphi = K_a \varphi',\) let \((u, \sigma', c')\) be such that \((w, \sigma, c)R_a(u, \sigma', c')\) and \(M \otimes A, (u, \sigma', c') \models \varphi',\) i.e. \(\sigma'\) is true in \(u\) (see Remark 8). We have that \(wR_a u\), so by construction of \(M\), \((wR_a u) \in \Gamma\). Also, since \((w, \sigma, c)R_a(u, \sigma', c')\) we have that \((\sigma, c) \sim_a (\sigma', c')\). By saturation of rule \(K_a\phi\) we thus have that \((u, \sigma', c') \in \Gamma\), and by induction hypothesis \(M \otimes A, (u, \sigma', c') \models \varphi',\) which concludes.

If \(\varphi = \neg K_a \varphi',\) by saturation of rule \(\neg K_a\phi\) there exist \((\sigma', c') \sim_a (\sigma, c)\) and a world symbol \(u\) such that \(\Gamma\) contains \((u, \sigma', c') \neg \varphi')\) and \((wR_u u)\). It follows that \((w, \sigma, c)R_u(u, \sigma', c')\). We also have that \(M \otimes A, (u, \sigma', c') \models \varphi'\) (or in other words, \(\sigma'\) is true in \(u\)), otherwise the branch would be closed by rule \(\top\). Finally, by induction hypothesis, \(M \otimes A, (u, \sigma', c') \models \neg \varphi',\) and thus \(M \otimes A, (w, \sigma, c) \models \neg K_a \varphi'.\) ■

Proposition 18 If \(\varphi_0\) is \(A\)-satisfiable, then \((A, \varphi_0)\) is consistent.

Proof. Suppose that there is a pointed model \((M_0, w_0)\) such that \(M_0 \otimes A, (w_0, \epsilon, 0) \models \varphi_0\). We must prove that every tableau for \((A, \varphi_0)\) has an open branch.

We let \(W_T\) denote the set of world symbols appearing in a set of tableau terms \(\Gamma\). Such a set \(\Gamma\) is said to be interpretable if, first, it does not contain \(\bot\) and, second, there is an initial model \(M = (W, \{R_a\}_{a \in \text{AGT}}, \psi)\) and a mapping \(f : W_T \rightarrow W\) such that:
for each \((w_a e A)\) such that \(\Gamma_0 = \{(w_0 e 0 \varphi_0)\}\) does not contain \(\perp\), and by assumption there is a pointed model \((M_0, w_0)\) such that \(M_0 \otimes A (w_0, e, 0) = \varphi_0\). So \(M_0, [w_0 \rightarrow w_0] = \Gamma_0\), and \(\Gamma_0\) is interpretable. We now prove that when a tableau rule is applied in a node that carries an interpretable set of tableau terms and is not an end node, then one of its successors carries an interpretable set. This implies that every tableau for \((A, \varphi_0)\) has a branch whose end node carries an interpretable set; in particular, this set does not contain \(\perp\), so the branch is open, which concludes.

In the following, \(\Gamma\) is the interpretable set of tableau terms in which the rule is applied, and \(M = (W, \{R_a\}_{a \in AT}, w)\) and \(f : W_T \rightarrow W\) are such that \(M, f \models \Gamma\).

We treat the case of rules for propositional logic as it is straightforward.

Rule \(\text{ch}\) for atomic proposition \(p\), on numerator \(\{(w \sigma c \varphi)\}\): If \(p \in u(f(w))\), then \(M \otimes A, (f(w), e, 0) \models p\), and thus \(M, f \models \Gamma \cup \{(w e 0 p)\}\); otherwise \(M, f \models \Gamma \cup \{(w e 0 \neg p)\}\). So one of the successors is interpretable.

Rule \(O_\varphi\) on numerator \(\{(w \sigma c O_\varphi)\}\): by assumption, \(M \otimes A, (f(w), e, c) \models O_\varphi\). Thus, according to the semantics, we necessarily have that \(c(a) < |\sigma|\). So the only successor in the tableau carries the set \(\Gamma \cup \{(w \sigma c^+ \varphi)\}\). Since \(\{w \sigma c O_\varphi\} \in \Gamma\) and \(M, f \models \Gamma\), \(M \otimes A, (f(w), e, c) \models \varphi\), and thus also \(M \otimes A, (f(w), e, c^+) \models \varphi\). Besides, because \(M \otimes A, (f(w), e, c) \models O_\varphi\), we have that \(M \otimes A, (f(w), e, c^+) \models \varphi\). It follows that \(M, f \models \Gamma \cup\{(w \sigma c^+ \varphi)\}\), and the successor is interpretable.

Rule \(\text{\neg} O_\varphi\) on numerator \(\{(w \sigma c \neg O_\varphi)\}\): the application of this rule requires that \(c(a) < |\sigma|\) hold. So the fact that \(M \otimes A, (f(w), e, c) \models \neg O_\varphi\) holds implies \(M \otimes A, (f(w), e, c^+) \models \neg \varphi\). The consistency aspect is treated like for rule \(O_\varphi\), and we obtain that \(M, f \models \Gamma \cup \{(w \sigma c^+ \neg \varphi)\}\), hence the successor is interpretable.

Rule \(\langle \psi \rangle_{\text{async}}\) on numerator \(\{(w \sigma c \langle \psi \rangle_{\text{async}})\}\): We have that \(M \otimes A, (f(w), e, c) \models \langle \psi \rangle_{\text{async}}\), so \(\sigma;\psi \in \text{Seq}(A)\), which implies that it is the first version of the rule that is applied. We also have that \(M \otimes A, (f(w), e, c) \models \psi\) and \(M \otimes A, (f(w), e, \psi; c) \models \varphi\). From the former and the fact that \(M, f \models \Gamma \{(w \sigma c \langle \psi \rangle_{\text{async}})\}\), we obtain that \(M \otimes A, (f(w), e, \psi; c) \models \varphi\). It follows that \(M, f \models \Gamma \cup \{(w \sigma c \psi; c)\}\), and thus the only possible successor is interpretable.

Rule \(\neg \langle \psi \rangle_{\text{async}}\) on numerator \(\{(w \sigma c \neg \langle \psi \rangle_{\text{async}})\}\): First, because \(\{(w \sigma c \neg \langle \psi \rangle_{\text{async}})\} \in \Gamma\), we have \(M \otimes A, (f(w), e, c) \models \varphi\). Also, the application of this rule requires that \(\sigma;\psi \in \text{Seq}(A)\). So the fact that \(M \otimes A, (f(w), e, c) \models \neg \langle \psi \rangle_{\text{async}}\) holds implies that either \(M \otimes A, (f(w), e, c) \models \neg \psi\) or \(M \otimes A, (f(w), e, \psi; c) \models \neg \varphi\). If \(M \otimes A, (f(w), e, c) \models \neg \psi\), we obtain that \(M, f \models \Gamma \cup \{(w \sigma c \neg \psi)\}\), and the first successor is interpretable. Otherwise we have both \(M \otimes A, (f(w), e, \psi; c) \models \neg \varphi\) and \(M \otimes A, (f(w), e, c) \models \psi\). The latter implies that \(M \otimes A, (f(w), e, \psi; c) \models \varphi\); we obtain that \(M, f \models \Gamma \cup \{(w \sigma c \psi; c)\}\), and the second successor is interpretable.

Rule \(K_a\varphi\) on numerator \(\{(w \sigma c K_a\varphi)\}\), for some \((\sigma', c') \sim_a (\sigma, c)\) and \(\sigma'\) true in \(u\). First, since rule \(\text{ch}\) has the priority over rule \(K_a\varphi\), we know that \(\Gamma\) is saturated for rule \(\text{ch}\). Also, since \((w R_a u) \in \Gamma\) and tableau terms of this form can only be introduced by rule \(\neg K_a\varphi\) together with a tableau term of the form \((u \sigma'' c'' \varphi'')\), then there is one such tableau term in \(\Gamma\). By saturation of rule \(\text{ch}\), it follows that for each \(p\) appearing in \(\varphi_0\) and \(A\), either \((u e 0 p)\) or \((u e 0 \neg p)\) is in \(\Gamma\). This defines a valuation \(\nu\) for \(u\) that, by assumption, makes \(\sigma'\) true (see Remark 5). Because \(M, f \models \Gamma\), we have that \(u(f(u))\) agrees with \(\nu\) on all atomic propositions in \(A\).
and therefore \( M \otimes A, (f(u), \sigma', c') \vDash \varphi \). Now, since \( M, f \models \Gamma \) and \((wR_0 u) \in \Gamma\), we have that \( f(w)R_0 f(u) \), and because \((w \sigma c K_u \varphi) \in \Gamma\), it holds that \( M \otimes A, (f(w), \sigma, c) \models K_u \varphi \). Since \( f(w)R_0 f(u) \) and \((\sigma, c) \sim_a (\sigma', c') \), we have that \( (f(w), \sigma, c)R_0 (f(u), \sigma', c') \). As we have seen that \( M \otimes A, (f(u), \sigma', c') \vDash \varphi \), we finally have that \( M \otimes A, (f(u), \sigma', c') \models \varphi \), thus \( M, f \models \Gamma \cup \{(u \sigma' c' \varphi)\} \), and the successor is interpretable.

Rule \( \neg K_u \varphi \) on numerator \( \{(w \sigma c \neg K_u \varphi)\} \): since \( M \otimes A, (f(w), \sigma, c) \models \neg K_u \varphi \), there exist \( u \in W, \sigma' \) and \( c' \) such that \((f(w), \sigma, c)R_0 (u, \sigma', c'), M \otimes A, (u, \sigma', c') \models \varphi \) and \( M \otimes A, (u, \sigma', c') \models \neg \varphi \). Recall that \((f(w), \sigma, c)R_0 (u, \sigma', c') \) means that \( f(w)R_0 u \) and \((\sigma, c) \sim_a (\sigma', c') \). Clearly, \( M, f[u \mapsto u] \models \{(u \sigma' c' \neg \varphi)(wR_0 u)\} \), and because \( u \) is fresh, \( f[u \mapsto u] \) coincides with \( f \) on all world symbols appearing in \( \Gamma \), so that \( M, f[u \mapsto u] \models \Gamma \). Finally, the denominator corresponding to \( \sigma', c' \) is interpretable (there are only finitely many possible \( \sigma' \) and \( c' \), see proof of Theorem 21).

Rule \( \varphi \) on numerator \( \{(w \sigma c \varphi)\} \): because \( \Gamma \) is interpretable, this rule cannot be applied. Indeed, assume it is applied. Because rule \( ch \) is applied in priority, \( \Gamma \) is saturated for rule \( ch \). With reasoning similar to that followed for rule \( K_u \varphi \), we obtain that the valuation \( \nu \) defined by \( \Gamma \) for \( w \) coincides with \( \psi(f(w)) \) on all atomic propositions appearing in \( \varphi_0 \) and \( A \), and thus they agree on all formulas in \( \sigma \). Yet on the one hand, since \((w \sigma c \varphi) \in \Gamma \) and \( M, f \models \Gamma \), we have that \( M \otimes A, (f(w), \sigma, c) \models \varphi \) and thus \( \psi(f(w)) \) satisfies all formulas in \( \sigma \). On the other hand, because the rule \( \varphi \) is applied, \( \nu \) does not satisfy all formulas in \( \sigma \), and we have a contradiction.

Rule \( \bot \) on numerator \( \{(w \epsilon 0 p), (w \epsilon 0 \neg p)\} \): because \( M, f \models \Gamma \) this cannot happen, as otherwise we would have both \( p \in \psi(f(w)) \) and \( p \notin \psi(f(w)) \).

**Theorem 21** The satisfiability problem for finite propositional protocols is in \( \text{NExptime} \).

**Proof.**

Let \( A \) be a propositional and finite protocol and \( \varphi_0 \) be the formula to check. The algorithm to check whether \( \varphi_0 \) is \( A \)-satisfiable consists of non-deterministically applying tableau rules of Figure 9.5 from the initial tableau \( \{(w \epsilon 0 \varphi_0)\} \).

Each world symbol \( w \) except \( w_0 \) is created by rule \( \neg K_u \) with a formula \( \varphi_w \), and the number of times this rule is applied to terms with \( w \) as world symbol is linear in \( \varphi_w \). These world symbols can be ordered in a tree structure (a world symbol created by applying rule \( \neg K_u \) in a tableau term with world symbol \( w \) is a child of \( w \)), and the modal depth of \( \varphi_w \) formulas is strictly decreasing in the tree. So the number of created world symbols \( w \) is exponential in the size of \( \varphi_0 \).

In addition, recall that the number of possible sequences of announcements \( \sigma \) is exponential in the size of \( A \), and the number of possible cuts \( c \) is \(|A|^{\lceil AGT \rceil}\). Therefore, the number of different tableau terms \( (w \sigma c \psi) \) is exponential in \(|\varphi_0| + |A|\).

At each step, the algorithm is executing a rule that adds at least one term. As the number of terms is exponential, the number of rule applications is exponential, and thus the running time of the (non-deterministic) algorithm is exponential. So the satisfiability problem when the protocol is finite and propositional is in \( \text{NExptime} \).

We now establish the matching lower bound.

**Theorem 22** The satisfiability problem for finite propositional protocols with at least two agents is \( \text{NExptime} \)-hard.

**Proof.**

See in [KMS17].

### 9.7 Related work

We review several research areas related to different aspects of the present work.
9.7. RELATED WORK

9.7.1 Existing logics for asynchrony

As far as we know, there has not been much work on the relationship between knowledge, announcements and asynchrony. In [DLW11], asynchrony in dynamic epistemic logic is studied, with the notion of asynchrony being that an agent cannot tell whether an event has occurred if her epistemic state is unchanged. This notion of asynchrony is different from the one we consider in this work: indeed in [DLW11], different agents can have a different idea of how many events have occurred so far, because some events might be completely unnoticed by some agents. So in one setting asynchrony is due to events being completely unobserved, while in the other (the one considered in this work) it is due to a delay between the occurrence of an event (the announcement) and its observation (the reception).

A logic dealing with knowledge and asynchrony is also developed in [PT92], but in this setting, messages do not have logical content: for example, the logic does not allow for announcements about knowledge or about the effect of other announcements. [FRV92] is concerned with knowledge in multi-agent, dynamic systems which may be asynchronous, but does not explicitly model communication, and in particular the effects of asynchronous sending and receiving of true announcements, which is the focus of our work.

Recently, van Ditmarsch developed a logic of asynchronous announcements in [vD17]. The major difference between our framework and that one is our third basic principle, that agents are able to imagine all possible pending messages. In van Ditmarsch’s work, agents in fact do not consider any pending or future announcements possible; they only consider a message possible after they have received it. So in our work, an agent has three sources of uncertainty: first, uncertainty about the state arising from the underlying Kripke model; second “past uncertainty,” that is, uncertainty about which of the messages that has received have already been received by other agents; and third, “future uncertainty,” uncertainty about what messages are pending in the channel but unread by a, or which messages may be broadcast in the future. In van Ditmarsch’s work, agents only have the first two sources of uncertainty: uncertainty arising from the underlying Kripke model, and “past uncertainty.” This means that agents may not consider the current state possible, and may even have false knowledge. For example, if agents a and b initially do not know true proposition p, and then p is broadcast, in van Ditmarsch’s framework, if a has received broadcast p and b has not, b considers it impossible that a knows p, even though a does indeed know p. Symbolically, $K_a p \land K_b \neg K_a p$. In our logic this is not the case: even when b has not received the broadcast of p, b considers it possible that p has been broadcast and received by a, so the formula $K_a p \land K_b \neg K_a p$ can never hold in our models. In general, $K_a \varphi \rightarrow \varphi$ holds in our logic, while this is not the case in van Ditmarsch’s logic.

9.7.2 Semi-private announcements and dynamic epistemic logic

On first glance, asynchronous broadcast logic has some similarities with semi-private announcement logic [GG97, BMS98b, BM04, BDM08]. Logics with semi-private announcements follow the same basic idea as public announcement logic, but rather than announcements being received by the entire group of agents, each message is announced to a subset of agents, while the rest of the agents know the message was announced to that group, but do not know what the content of the message was. In the general setting, group A receives message m and updates their knowledge accordingly, while the agents not in group A know that A received either m or its negation, $\neg m$, and update their knowledge accordingly. The identity of the group receiving each message is common knowledge for everyone. On the surface, this logic has some similarities with asynchronous broadcast logic: at any time, a certain group of agents has received each message, while others have not. However, like other variants of public announcement logic, logics of semi-private announcements are synchronous: a message is sent and received simultaneously,
and thus common knowledge is achieved immediately by the group of agents receiving the message. Furthermore, even the group of agents who do not receive the message have synchronous information, since they immediately know that the other agents have received some message. Overall, in this setting, the agents have less uncertainty about one another’s knowledge than in the asynchronous setting. In fact, the issues of semi-private messages and asynchrony are orthogonal; one could imagine an asynchronous logic of semi-private announcements, where each member of group $A$ eventually receives announcement $m$, and the rest of the agents eventually receive the information that group $A$ has been asynchronously sent either $m$ or $\neg m$.

### 9.7.3 Arbitrary public announcement logic

Arbitrary public announcement logic (APAL) [BBvD+07] has some similarities to our approach. In this logic, one can ask whether some formula holds after any possible announcement; this is not possible in our logic, but because agents can imagine pending messages, our knowledge operator considers any possible future sequence of announcements that follows the protocol, which is a related idea. Interestingly, the satisfiability problem for APAL is undecidable, but decidability can be achieved by considering a constraint similar to our restriction to existential announcements [FvD08, vDFH].

### 9.7.4 Distributed systems

The systems we consider are closely related to the notion of total order broadcast in distributed systems [Ray13, p. 154]:

1. if a message is received, then it means that it has been broadcast;
2. no message is received twice;
3. if an agent received $\varphi$ before $\varphi'$, they all receive $\varphi$ before $\varphi'$;
4. $\varphi$ causally precedes $\varphi'$ implies that no agent receives $\varphi'$ before $\varphi$;
5. if a message is broadcast, all agents will eventually receive it.

The first point holds in our system since a message (a formula) is only received if it is in the queue, which is the list of broadcast messages. The second point holds because a message is received when an agent’s cut is increased to include that message from the queue, which only occurs once for each message. The third point holds because we have FIFO channels, and thus agents all receive messages in the same order, the order in which they are announced. The fourth point follows from the fact that in our systems we only consider a state $(w, \sigma; \psi, c)$ consistent if $(w, \sigma, c) \models \psi$, and because messages are received in order. The fifth point is not directly modelled in our systems since we only consider finite histories, but it is a kind of liveness constraint that we will probably be led to consider when we extend the logic with temporal operators (see next section).

More recently, [GL13] studies the model checking of distributed systems with respect to epistemic specifications. Although this work is in a synchronous setting, it is quite close to our approach in spirit, and shows that epistemic issues in distributed systems have practical implications, and a logical approach to these concerns can be fruitful.

Finally, we note that our definition of asynchronous models $M \otimes A$, especially the notion of cuts, is in the spirit of [Lam78].
9.8 Future work

This work is a first attempt to develop an epistemic logic for reasoning about asynchronous announcements. In the future, we would like to overcome the circularity problem, and define the semantics for the most general case (removing the finite tree and existential conditions). Using coinduction to define the set of consistent states may be one approach to this problem. Once we have defined the semantics for the general case, if possible, we hope to provide a complete axiomatization and a general model-checking algorithm. We also plan to implement the model-checking algorithms. Actually, we believe that the model checking of our logic could be reduced to recently proposed succinct languages for Dynamic epistemic logic [CS15, CS17]. Therefore, we could use symbolic techniques as presented in [vBvEGS15b].

Second, we would like to model more general situations of asynchronous communication. We plan to consider the case where messages are not read in FIFO order, but are received and read in arbitrary order. We also plan to model the origin of the messages, allowing formulas such as “After agent \(a\) broadcasts \(\varphi\), \(\psi\) holds”. In our current setting, when the external broadcaster makes a new announcement, the only effect is to queue it in the channel without affecting anyone’s epistemic state. However, in the case where the agents themselves make the announcements, agent \(a\) making an announcement should impact her knowledge: after the announcement she should know, for instance, that the channel is not empty. She should also know that after another agent checks their channel, that agent will know that \(\psi\) has been announced.

Third, it would be interesting to add temporal operators to our language, in order to express properties like “After \(p\) is announced and agent \(a\) receives it, eventually she will know that agent \(b\) knows \(p\)” (assuming that agents are forced to read announcements eventually).

Finally, we would like to model not only asynchronous broadcasts on a public channel but also private asynchronous communications between agents in the system. In essence, this amounts to defining a complete asynchronous version of dynamic epistemic logic [Van07].

Acknowledgements We would like to thank Hans van Ditmarsch who hosted the three authors in Nancy and who was the initiator of studying asynchrony in DEL.
Bibliography


[vDFH] Hans van Ditmarsch, Tim French, and James Hales. Positive announcements (under submission).


iterated deletion of strictly dominated strategies, 20

KD45, 18 104
Knaster-Tarski theorem, 105
knowledge base, 11
knowledge-based program, 9
knowledge-based programs, 10

learning, 29
linear mu-calculus, 78

mAL, 28
Metel2, 59
model checking, 43 88 108
monadic second-order logic, 69
mu-calculus, 78
muddy children, 15 33
multi-agent system, 7
multi-robot system, 10

negative introspection, 18
no forgetting, 25
node, 50

PDDL, 28
perfect recall, 25
pointed epistemic model, 15
positive introspection, 18
principle of inertia, 94
private announcement, 22
propostion, 15
psychology, 10
public action, 22
public announcement, 22 106
purely epistemic, 22

quotient, 89

ramification problem, 28
reachability problem, 11
real world, 15
reflexivity, 18
regular automatic tree, 73
relational structure, 69
robot, 5 10
rule 110, 62

S5, 18 64
Sally and Anne, 10 23
satisfiability problem, 49 111

seeing, 87
semantics, 20
semi-private announcement, 24
seriality, 18
social intelligence, 18
state, 15
STIPS, 28
synchronicity, 25
syntactic fragments, 29
syntax, 19

T, 18
tableau method, 49 111
theorem proving, 41
timely common knowledge, 21
total order broadcast, 113
transition system, 28
transitivity, 18
tree, 69
truth condition, 20
undistinguishability, 18
uniform strategy, 10
unraveling, 15

video game, 10