Conditional Epistemic Planning

Mads Krogsgaard Pico Jensen

Kongens Lyngby 2013
IMM-M.Sc.-2013-27

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2013-27

Summary (English)

Dynamic epistemic logic reasons about higher-order knowledge in multi-agent systems, and has shown promise as a framework for use in automated planning. Using dynamic epistemic logic in automated planning allows for capturing of partial-observability and non-determinism, making it an highly expressive combination. This thesis is based on the later work within this field by Andersen, Bolander, and Jensen [1], wherein a planning algorithm is proposed for finding strong and weak plans in a single-agent setting. Their planning algorithm takes the internal view of the agent, necessary for developing planners for use in autonomous agents.

The main contribution made by this thesis is a planning algorithm for finding strong cyclic and weak plans in a multi-agent setting with one acting agent using dynamic epistemic logic. The algorithm takes the internal perspective of the acting agent, thereby making it the planning agent. The planning algorithm is shown to be both sound and complete, and the work is completed by an implementation of the proposed planning algorithm.

Summary (Danish)

Dynamiske epistemiske logik giver mulighed for at ræsonnere om højere ordens viden i multi-agent systemer, og har vist potentiale til bruge i automatiseret planlægning. Automatiseret planlægning der gør brug af dynamisk epistemisk logik kan indfange delvis-observerbarhed og ikke-determinisme, hvilket gør det til en yderst udtryksfuld kombination. Denne afhandling er baseret på det senere arbejde af Andersen, Bolander, og Jensen [1] inden for dette område, hvori de viser en planlægningsalgoritme der kan finde stærke og svage planer i et system med én agent. Planlægningens algoritmen planlægger med perspektivet fra agenten, hvilket er nødvendigt hvis målet er at udvikle planlæggere til anvendelse i autonome agenter.

Hovedbidraget i denne afhandling er en planlægningsalgoritme til at finde stærkcykliske og svage planer i et multi-agent system, med én handlende agent ved bruge af dynamisk epistemisk logik. Algoritmen tager det interne perspektiv af den handlende agent, og gør derved agenten til den planlæggende agent. Planlægningens algoritmen er vist både sund og fuldstændige, og arbejdet er afsluttet med en implementation af algoritmen.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modeling at the Technical University of Denmark from October 2012 to April 2013 under supervision of associate professor Thomas Bolander and PhD-student Mikkel Birkegaard Andersen. It has an assigned workload of 35 ECTS credits.

Acknowledgements. I would like to thank both of my supervisors for their help and advice during the completion of the thesis.

Mads Krogsgaard Pico Jensen

Kgs. Lyngby, April 2013

Contents

Chapter 1

Introduction

The field artificial intelligence within computer science is the study of intelligent (or rational) agents. An agent is one who perceives its environment and acts upon it, and a rational agent acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome [11].

Automated planning is a subfield within artificial intelligence concerned with reasoning about actions. One purpose of automated planning is to enable agents to behave rationally by reasoning about their actions prior to acting. The complexity of this reasoning depends on the environment that the agent is situated in and the goal it must accomplish. Non-classical planning deals with environments that resemble realistic (real life) environment. Such environments may contain multiple acting agents, acts whose precise outcome cannot be known (non-determinism), and agents with only partial-knowledge about the state of the environment (partial-observability).

When presented with uncertainty a course of action for achieving a goal is not necessarily a sequence of actions, but a conditional plan that specifics what to do depending on what percepts are received from the environment. If an act can have two possible outcomes, a conditional plan can specific different course of action for each. When the actual outcome of the act is known, the agent can continue with the appropriate course of action.

Research within non-classical planning includes autonomous agents that do not require complete knowledge of their environment, but can reason about how to gain any missing knowledge necessary to achieve their goal. The concept is illustrated in the following example.

Example 1.1. *Bob is on his way to a surprise birthday party for Alice at her place, when he suddenly realize that he has left Alice's address at home. Bob does not have time to go back and get it, but he is not concerned because he knows he can call Jack to get the address.*

Bob is in a situation where he must know the address of Alice to achieve his goal, and so he plans a course of action for obtaining this missing piece of information. The scenario and line of reasoning is normal in everyday life and something that can be expected of a rational agent.

For Bob to reason as he did, he must not only have knowledge about the state of the environment, he must also have knowledge about what Jack knows. This type of knowledge is called higher-order knowledge and plays an important role in human interaction, which is the motivation behind in-cooperating it into a rational agent. One of the promising logical frameworks for modeling and capturing higher-order knowledge is dynamic epistemic logic (DEL).

Motivation

The motivation behind this thesis is to build upon some of the recent work done with combining automated planning with dynamic epistemic logic [1][3][2]. The work shows dynamic epistemic logic as a solid foundation for automated planning within an partial-observability and non-determinism environment.

Outline of this thesis

Chapter 2 introduces the elements of dynamic epistemic logic used in this thesis and cast them in a planning setting.

Chapter 3 defines the epistemic planning problem for planning using dynamic epistemic logic, the solutions types for it, and how plans are represented.

Chapter 4 introduces and defines the proposed planning algorithm, and describes how it differs from the one proposed by Andersen, Bolander, and Jensen [1].

Chapter 5 contains the proofs for soundness and completeness of the planning algorithm.

Chapter 6 describes the implementation of the planning algorithm.

Chapter 7 concludes upon the work done in this thesis.

Resources

The main resources for each chapter will be noted in the introduction for the chapter. If a statement (definitions, theorems, lemmas, and examples) is not the work of this thesis, then an [X] showing the bibliographic reference to the source will be contained withing the name of the statement.

Notation

The thesis makes use the following notations

AND/OR–graph	An AND/OR-graph G is defined by $G = (V, E)$, where V is the nodes in G composed of the OR-nodes V_{OR} and AND-nodes V_{AND}. E is the edges connecting nodes.
Path	A path $n \rightarrow_G m$ in a graph G (or just $n \rightarrow m$ if G is clear from the context) leads from the node n to the node m using a single edge. The path $n_1 \rightarrow^* n_k$ leads from node n to n_k, such that $n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_k$, and the nodes in the path are the set $\{n_1, n_2, ..., n_k\}$. A path is non-cyclic e.g. no nodes is visited twice.
Subgraph	A subgraph G_n at node n in G is a graph where the node n and all nodes m for which there exists a path $n \rightarrow^*_G m$ are in.
Partial subgraph	A partial subgraph H of G is a graph whose nodes are a subset of that of G, and whose edges are connected such that if a path $n \rightarrow^*_H m$ exist, then the same path exists in G.
Depth	The depth of a node m with respect to a node n is defined as the longest path from n to m.
Height	The height of a node n is the longest path from n to a leaf node.

Chapter 2

Dynamic Epistemic Logic

This chapter introduces the part of dynamic epistemic logic used throughout the thesis. The main resources for the chapter are [1], [3], and [6], and most of the definitions in this chapter are from these. The only contribution made in this chapter is that of internal states in a multi-agent setting. The concept is defined by Andersen, Bolander, and Jensen [1] for the single-agent setting, but it has been necessary to extend it for the multi-agent setting in the later planning.

The chapter first introduces a toy example *The Prisoner and the Guard*, which will be used through out the thesis to illustrate the different concepts as they are introduced.

2.1 The Prisoner and the Guard

Example 2.1 (The Prisoner and the Guard). *The prisoner has just arrived to the county prison, and is going to spend some time there for his crimes. However, the prisoner has, using his lock picking skills, been able to unlock the cell door, and is now planning his escape. The prisoner knows from previous visits that from his cell he only needs to get to the door at the end of the hallway, which is known never to be locked, and behind it lies freedom.*

There is only one problem, the guard is watching the exit.

Right now, the guard is facing the exit, and the proposition `f` *will be used to denote this fact. The prisoner can still make a run for the outer door, and is sure to escape, denoted* `e`*, if he does so. However, the prisoner knows that if the guard knows he has escaped, then he will not be free for long before being recaptured. The prisoner therefor considers two options before running for the exit.*

The first is to try and harass the guard until he gets annoyed and goes for a donut.

The second option is to bribe the guard. It is well known that some guards are corrupt and can be bribed to look the other way, but whether this particular guard is corrupt or not is unknown to the prisoner. In the following, `c` *will be used to denote that the guard is corrupt, and* `b` *to the denote that the guard has accepted a bribe.*

The symbol `g` *is used for the guard and* `p` *for the prisoner. For the prisoner to successfully escape (with a chance to stay free), he must either escape without the guard knowing, or escape after having bribed the guard.*

2.2 Modeling the State of Affairs

2.2.1 Epistemic Language

The following will introduce the language of multi-agent epistemic logic.

Definition 2.1 (**Epistemic Language [3]**). *Let $\mathcal{P}$ be a finite set of propositional symbols, and $\mathcal{A}$ a finite set of agents. The language of multi-agent epistemic logic $\mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ on $(\mathcal{P}, \mathcal{A})$ is the formulas generated by the following BNF:*

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid K_i\phi$$

where $\top$ is the truth symbol, $p \in \mathcal{P}$, $i \in \mathcal{A}$, ϕ is an epistemic formula, and the $K_i\phi$ is the knowledge operator that should be read as: agent i knows the formula ϕ holds.

Given the definition above, the goal ϕ_p of the prisoner, with $\mathcal{P} = \{b, c, e, f\}$ and $\mathcal{A} = \{g, p\}$, can be expressed by the epistemic formula

$$\phi_p = e \wedge (b \vee \neg K_g e). \tag{2.1}$$

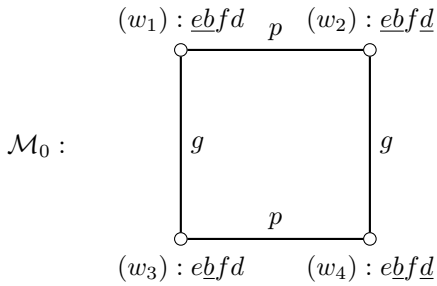


Figure 2.1: The epistemic model for the initial situation in example 2.1 on page 5. The reflexive relation for each agent at each world is left out.

The formula states that the prisoner should have escaped (e) after he bribed (b) the guard, or the prisoner should have escaped (e) with the guard not knowing he has escaped ($\neg K_g e$).

2.2.2 Possible Worlds

Dynamic epistemic logic models knowledge using a concept of possible worlds. A possible world is one that contains one state of affairs that the agents consider possible. To model the *uncertainty* that agents have about the state of affairs, a model of their knowledge can contain multiple possible worlds with different state of affairs. Such a model is called a *kripke structure*, or an *epistemic model*.

Definition 2.2 (**Epistemic Models [3]**). *An epistemic model on the language $\mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ is a triple $\mathcal{M} = (W, R, V)$ where*

- *W is a non-empty finite set of (possible) worlds*[1]*,*
- *$R : A \to 2^{W \times W}$ assigns an equivalence relation*[2] *(called an indistinguishability relation) R_i, to each agent $i \in \mathcal{A}$,*
- *$V : P \to 2^W$ assigns to each proposition $p \in \mathcal{P}$ the set of world in which p holds.*

The initial situation in example 2.1 on page 5 can be modeled using the epistemic model $\mathcal{M}_0$ shown in figure 2.1.[3] The model consists of the four worlds $\{w_1, w_2,$

[1] In the remaining of the thesis a possible world will just be name a world.
[2] An equivalence relation is reflexive, symmetric and transitive
[3] The indicsinguishability relation will be shown without the reflexive loops in the reminder of this thesis.

w_3, w_4}, where in each world, the proposition symbols that are not true are underlined. An edge between two worlds means that the labeled agent(s) cannot *distinguish* the two worlds apart, and are therefore uncertain about which one models the *actual* state of affairs.

In the model, the prisoner does not know whether the guard is corrupt or not, and considers both options possible. The guard on the other hand does not know if the prisoner has escaped or not, but he will only raise the alarm when he is certain. Both the guard and the prisoner know with certainty that the guard is facing the exit, and that the guard has not accepted a bribe from the prisoner.

Definition 2.3 (**Truth in Epistemic Models [3]**). *Let $\mathcal{M} = (W, R, V)$ on $\mathcal{L}_\mathcal{K}(\mathcal{P}, \mathcal{A})$ be an epistemic model, and let $i \in \mathcal{A}$, $w \in W$, $p \in \mathcal{P}$, and $\phi, \phi_1, \phi_2 \in \mathcal{L}_\mathcal{K}(\mathcal{P}, \mathcal{A})$, then the truth of a formula in w is defined by*

$\mathcal{M}, w \models \top$	*always*
$\mathcal{M}, w \models \bot$	*never*
$\mathcal{M}, w \models p$	*iff* $p \in V(w)$
$\mathcal{M}, w \models \neg\phi$	*iff* $\mathcal{M}, w \not\models \phi$
$\mathcal{M}, w \models \phi_1 \wedge \phi_2$	*iff* $\mathcal{M}, w \models \phi_1$ *and* $\mathcal{M}, w \models \phi_2$
$\mathcal{M}, w \models K_i\phi$	*iff* $\mathcal{M}, v \models \phi$ *for all* $v \in W$ *where* wR_iv

2.2.3 Actual Worlds

Even though an epistemic model $\mathcal{M}$ can contain multiple possible worlds, only one of these can model the actual state of affairs. Let $\mathcal{M} = (W, R, V)$ and $w \in W$ be this actual world, then the pair $(\mathcal{M}, w)$ is named a pointed epistemic model. Identifying a single actual world is often an ability that only an *external observer* has, e.g. one who is looking at the world from the *outside*. When the view of the state of affairs is not that of an external observer, but that of an non-omniscient agent in the world, then the model should reflect that as explained below.

From the point of view of an agent standing in the world, it is often not possible to tell exactly which world is the actual world, but it can instead be narrowed down to a set of actual worlds[4], then the pair $(\mathcal{M}, W_d)$ is a multi-pointed epistemic model. Multi-pointed epistemic models will through the remaining of the thesis be named

[4]This set is called *designated world* by Bolander and Andersen [3].

epistemic states, or just *states*, and denoted $\mathbf{s}$. The actual worlds in a state will be marked by double circles.

Definition 2.4 (**Truth in Epistemic States [3]**). *Let $\mathcal{M} = (W, R, V)$ on $\mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ be an epistemic model, let $(\mathcal{M}, W_d)$ be an epistemic state with $W_d \subseteq W$, and let $\phi \in \mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ then*

$$(\mathcal{M}, W_d) \models \phi \qquad \text{iff } (\mathcal{M}, w) \models \phi \text{ for all } w \in W_d$$

Returning to the prisoner and guard example, the prisoner knows he has not yet escaped and is therefore able to narrow the actual worlds down to the two worlds w_1 and w_2. Thus, the epistemic state modeling the initial situation from the view of the prisoner is $\mathbf{s}_0 = (\mathcal{M}_0, \{w_1, w_2\})$. This way of modeling the view of a situation from the perspective of a specific agent will be elaborated in section 2.4 on page 12.

2.3 Changing the State of Affairs

2.3.1 Possible Events

An act can result in physical changes in the environment (ontic change), or changes in the knowledge of agents in the environment, or both. Dynamic epistemic logic uses the concept of a *possible event* to describe the outcome of an act that an agent considers possible. To capture the uncertainty an agent has about the outcome of an act, a model of its knowledge about an act can contain multiple possible events. Such an model is called an *event model*.

Definition 2.5 (**Event Models [3]**). *An event model on $\mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ is defined by $\mathcal{E} = (E, Q, pre, post)$, where*

- *E is a non-empty finite set of (possible) events*[5]
- *$Q : 2^{E \times E}$ assigns an equivalence relation (called the indistinguishability relation) to each agent $i \in \mathcal{A}$.*
- *$pre : E \to \mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ assigns to each event a precondition.*
- *$post : E \to \mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ assigns to each event a postcondition.*

An event model describes which changes, both ontic and epistemic, an act causes. The set of events E are the events that an agent considers possible

[5] In the remaining of the thesis a possible event will just be name an event.

when performing the act. An edge between two events symbolizes that the labeled agent(s) cannot distinguish the two events apart, and are not able to tell which of them happened.

An example is that of a coin toss, which can result in the coin landing on either head or tails. The act of flipping a coin would be modeled by two events, one where the post-condition was tails, and the other that of head. The precondition would in both events be that a coin is available to toss.

If the coin lands on a table and is thereby visible after the toss, then the agent would know the outcome of the act, e.g. the agent would be able to tell the two events apart. The event would therefore not be connected by an agent relation. If the coin instead was covered by a hand, then the agent would know that the coin was tossed, and that it has either landed on heads or tails, but not which. In this case, the act would be modeled by the two events being connected.

2.3.2 Actual Events

As with the epistemic model, there will always be only one actual event happening. Let e be this actual event, the tuple $(\mathcal{E}, e)$ is then a pointed event model. However, when modeling an event from the perspective of a specific agent, it is not always possible to narrow it down to one actual event, but instead to a set of actual events E_d. Such an event model $(\mathcal{E}, E_d)$ is a multi-pointed event model, and will throughout the thesis be named an *epistemic action*, or *action*, and denoted `e`. The actual events in an epistemic action will be marked by double circles.

The actions available to the prisoner are shown in figure 2.2 on the next page. The prisoner can always harass the guard, making the precondition of the action `harass` $\top$. The action has two possible outcomes, where either the guard faces away from the cell $\neg f$, or he remains facing the same direction. Since the two events are not connected by any edge, both agents will know which event actually occurred after the act. The outcome of `bribe` depends on whether or not the guard is corrupt. Since the events b_1 and b_2 can be distinguished by the prisoner, he will afterwards know whether the guard is corrupt or not. Last `run` has two actual events r_1 and r_2 depending on whether the guard is facing the exit or not. The prisoner knows the last event r_3 will never occur, but even so the guard is not able to distinguish between this and r_2. r_2 and r_3 models that the prisoner knows that when the guard is not facing the exit, then the guard will be uncertain about whether the prisoner has escaped or not.

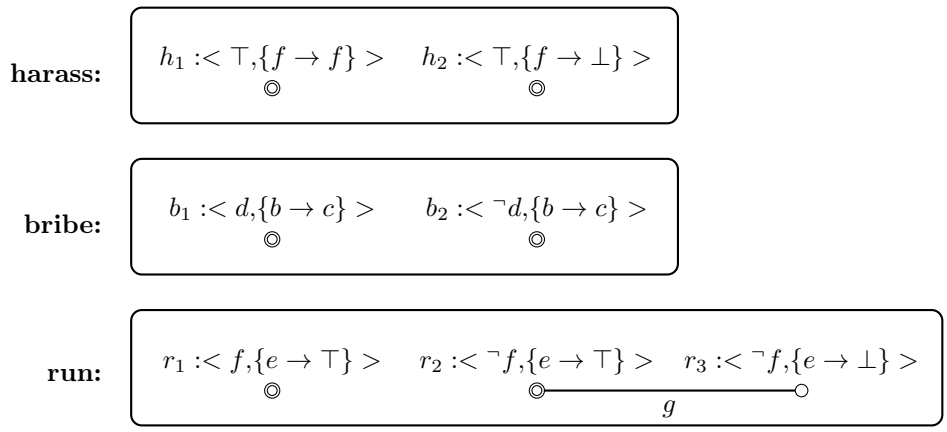


Figure 2.2: The actions available to the prisoner in example 2.1 on page 5

2.3.3 Acting

When an agent acts, it is modeled by *applying* an action to a state resulting in a new *updated* state. Dynamic epistemic logic incorporates this update using the *product update*.

Definition 2.6 (**Product Update [3]**). *The product update of a state* $(\mathcal{M}, W_d)$ *with an action* $(\mathcal{E}, E_d)$ *is defined as the state* $(\mathcal{M}, W_d) \otimes (\mathcal{E}, E_d) = ((W', R', V'), W'_d)$*, where*

$$\begin{aligned} W' &= \{(w, e) \in W \times E \mid \mathcal{M}, w \models pre(e)\}, \\ R'_i &= \{((w, e), (v, f)) \in W' \times W' \mid wR_iv \text{ and } eQ_if\}, \\ V'(p) &= \{(w, e) \in W' \mid \mathcal{M}, w \models post(e)(p)\} \text{ for each } p \in \mathcal{P}, \\ W'_d &= \{(w, e) \in W' \mid w \in W_d \text{ and } e \in E_d\}. \end{aligned}$$

The worlds remaining after a product update are those that satisfied the precondition of one or more events. To ensure that an action is not applied in a state where one of the actual worlds does not satisfies the precondition of any actual events, the following definition of applicability is used.

Definition 2.7 (**Applicability of an action in a state [3]**). *Let* $(\mathcal{M}, W_d)$ *be an epistemic state and* $(\mathcal{E}, E_d)$ *an action. Then* $(\mathcal{E}, E_d)$ *is said to be applicable in* $(\mathcal{M}, W_d)$ *if it holds that for each world* $w \in W_d$ *there is at least one event* $e \in E_d$ *such that* $\mathcal{M}, w \models pre(e)$.

The definition ensures that the knowledge about the actual worlds are consistent with the actual state of affairs, and a state is not reached wherein the set of

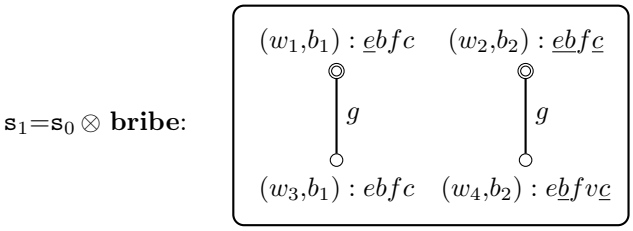


Figure 2.3: The resulting state from taking the product update between the initial state $\mathbf{s}_0$ and the action `bribe`

actual worlds are empty. The figure 2.3 shows the result when the prisoner from the initial situation considers what would happen if he tried to bribe the guard. The new worlds are named as the previous world with the applied event appended to it.

2.4 Planning with Dynamic Epistemic Logic

This section will first explain how dynamic epistemic logic relates to planning, and how partial-observability and non-determinism can be described by it. It will then introduce the internal perspective used to model the view of the world as seen from a specific agent (the acting agent). It is ended with a description of how to minimize and compare states.

2.4.1 Planning Environment

Imagine that the prisoner was not able to see the guard and would therefore not be able to know if the guard was facing the exit or not. Then the prisoner would not know which of the two possible outcomes of harassing the guard that actually occurred, and the two event h_1 and h_2 would have to be connected by an relation in R_p to properly model the act. This illustrates how partial-observable actions can be modeled. Moreover, the initial state can itself be partial-observable (prisoner does not know whether guard is corrupt or not). This shows how dynamic epistemic logic can be used for planning in a partial-observable environment.

Now consider the action of bribing the guard. There are two possible outcomes of this action, but which one that actually occurs depends entirely on whether the guard is corrupt or not, e.g. the outcome depends on the state that the

action is applied in. This makes this action deterministic[6]. However, the action of harassing the guard also has two outcomes, but with the same precondition. The actual outcome of this act is thus not dependent on the state of the world, but decided by the environment, which the agent has no control over. Dynamic epistemic logic it this way capable of modeling non-deterministic actions.

When the prisoner is planning his escape, he will do so by reasoning about the state of the world, the different actions at his disposal, and how they can be used to change the state of the world. Since it is the prisoner himself that is planning the escape, the models used should represent the view of the prisoner, and this is accomplished using *local states*, a concept introduced by Andersen and Bolander [3].

2.4.2 Local states

In definition 2.4 on page 9 a formula is said to be true in a state if and only if it is true in each of the actual worlds. When modeling the perspective of an agent, a formula should be true only when the agent knows it to be true. A local state for an agent is therefor a state wherein the agent can distinguish the set of actual worlds from the rest.[7]

The state $\mathtt{s}_0$ is an local state of agent p, but not of agent g, since the set of actual worlds are not closed under R_g.

Imaging that the prisoner considers bribing the guard, the resulting state would be $\mathtt{s}_1$ shown in figure 2.3 on the preceding page, and it is again a local state of the prisoner. The following lemma states that this will be the case in general.

Lemma 2.1 ([3]). *If a local action $(\mathcal{E}, E_d)$ of an agent i is applicable in a local state $(\mathcal{M}, W_d)$ of the same agent, then the product update $(\mathcal{M}, W_d) \otimes (\mathcal{E}, E_d)$ is again a local state of i.*

In the example in figure 2.3 on the facing page, the prisoner is now able to distinguish the two actual worlds. This means that even though the prisoner cannot tell which one will occur while reasoning about them, when the action is actually executed, the prisoner will know which one occurred and they are therefore *run-time distinguishable*.

[6] The result from bribing the guard in the initial state is shown in figure 2.3 on the preceding page.

[7] A *local action* of an agent is one wherein the agent can distinguish the actual events from the rest.

When two sets of actual worlds that are run-time distinguishable are encountered, it then allows the agent to reason about a separate plan of action for each set. This corresponds to a *conditional* plan. Each of these distinguishable sets can therefore be reasoned about separately. This then limits the necessary reasoning about a course of action to states where all the actual worlds are interconnected, and thereby both plan-time and run-time indistinguishable. Such states are called internal states of the planning agent[8] by Andersen, Bolander, and Jensen [1].

In the single-agent setting the constituting internal states of a local state[9] are found in a straightforward manner, by dividing the state into the minimum number of states where all world in a state are interconnected [1]. An approach for dividing a local state into internal states for the multi-agent setting are proposed in the following section.

2.4.3 Internal states

Dividing an local state into its constituting internal states are accomplished in two steps.

The first step is to divide the local state into states where all worlds are connected by *some* relation. These states are then named *global distinguishable* states, because all agents will able to distinguish them at runtime.

The state shown on the left in figure 2.4 on the next page is a local state of agent i and contains three actual worlds $\{w_1, w_2, w_3\}$. The states in the middle are the result of the division into global distinguishable states. In the example both of the two resulting states contain one or more actual world. However, assume w_2 was not an actual world, then the state are known by the planning agent to never be the case, and can be *removed*. To remove such a state is reasonable, because it no longer represent a possible state of affairs in this course of action.

[8] When the planning agent is clear from the context, the states will just be named internal states.

[9] Andersen, Bolander, and Jensen [1] uses only an epistemic model for modeling the state of affairs in the single-agent setting. This corresponds to all possible worlds being actual worlds.

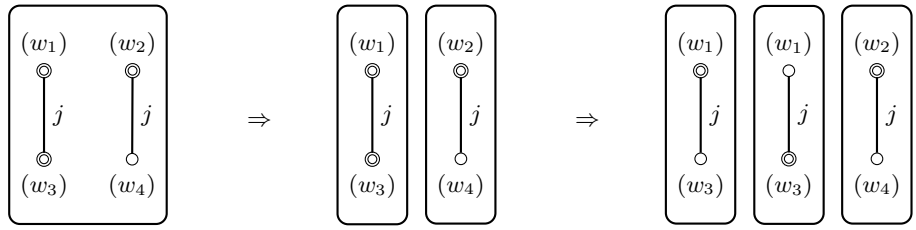


Figure 2.4: Division of a state with actual worlds $\{w_1, w_2, w_3\}$ into, first it constituting globally distinguishable states, and thereafter each of those it their constituting distinguishable state from the view of agent i.

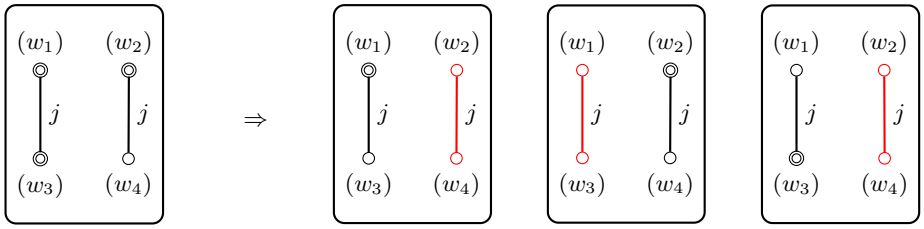


Figure 2.5: Division of a state with actual worlds $\{w_1, w_2, w_3\}$ into its constituting distinguishable state with actual worlds $\{w_1\}$, $\{w_2\}$, $\{w_3\}$ from the view of agent i.

The next steps is to divide each of the global distinguishable states into internal states. Let $(\mathcal{M}, W_d)$ be a local state of agent i, then the set of actual worlds are divided into minimum sets such that

1. there does not exists a relation in R_i between two worlds in different sets,
2. all worlds in each set are interconnected.

This will result in the sets $W_{d_1} \cup W_{d_2} \cup ... \cup W_{d_k} = W_d$ for some k where $1 \leq k \leq |W_d|$.[10] The agent will then be able to distinguish between each of these sets at run-time.

When considering each set of actual worlds, the epistemic model M should remain the same, so that the knowledge in the system stays the same. The resulting set of epistemic states will therefore be $\{(\mathcal{M}, W_{d_1}), (\mathcal{M}, W_{d_2}), ..., (\mathcal{M}, W_{d_k})\}$.

In figure 2.4, the result is the three states on the right, and the planning agent can now reason about each possible situation separately. If the first step is

[10] $|W_d|$ denotes the size of the set W_d

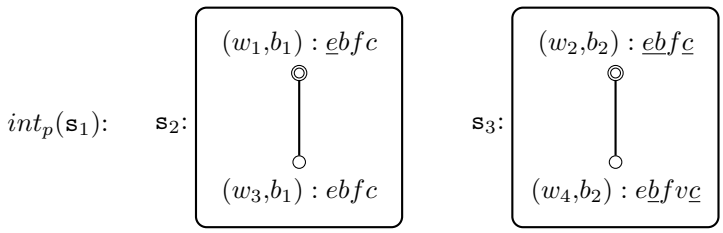


Figure 2.6: The two internal states $\mathtt{s}_2$ and $\mathtt{s}_3$ of agent p in the state $\mathtt{s}_1$.

left out, and the second step was done directly, then the result would be as illustrated in figure 2.5 on the previous page. Here each state now contains all nodes from the original, only differing on actual worlds. The set of nodes that are marked in red, are then redundant knowledge, representing a situation that the planning agent has considered in one of the other states.

These two steps for an agent i, and a local state $\mathtt{s}$ of i, will be denoted by $int_i(\mathtt{s})$. The result of dividing $\mathtt{s}_1$ into its constituting internal states is shown in figure 2.6, where the two possible outcomes of bribing the guard now are represented by their own state.

2.4.4 Comparing States

Two epistemic states can model the same knowledge without being isomorphism. Such two states will satisfy the same sets of epistemic formulas, and are considered equivalent because an agent will not able to distinguish between them.

Two states `s` and `s'` are said to represent the same knowledge if and only if there exists a bisimulation between them, denote by $\texttt{s} \leftrightarrows \texttt{s'}$, and are then named *bisimilar*.

The following is a standard definition of bisimulation between two epistemic models by Ditmarsch, Hoek, and Kooi [6]

Definition 2.8 (**Bisimulation between epistemic models [6]**). *Let two models $\mathcal{M} = (W, R, V)$ and $\mathcal{M}' = (W', R', V')$ be given. A non-empty relation $B \subseteq W \times W'$ is a bisimulation iff for all $w \in W$ and $w' \in W'$ with $(w, w') \in B$:*

atoms *$w \in V(p)$ iff $w' \in V'(p)$ for all $p \in P$*

forth *for all $i \in \mathcal{A}$ and all $w \in W$, if $(w, w) \in R_i$, then there is a $w' \in W'$ such that $(w', w') \in R'_i$ and $(w, w') \in B$*

back *for all $i \in \mathcal{A}$ and all $w' \in W'$, if $(w', w') \in R'_i$, then there is a $w \in W$ such that $(w, w) \in R_i$ and $(w, w') \in B$*

The concept is extended to multi-pointed epistemic models (states) by Bolander and Andersen [3].

Definition 2.9 (**Bisimulation between epistemic states [3]**). *Let $(\mathcal{M}, W_d)$ and $(\mathcal{M}', W'_d)$ be two epistemic states, an relation $B \subseteq W \times W'$ is then a bisimulation iff it satisfies the standard definition and further more satisfies that the domain of B extends W_d and the image of W_d under B is W'_d.*

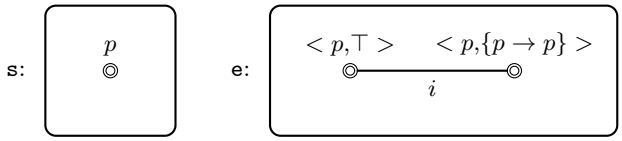


Figure 2.7: The epistemic state s and the action a

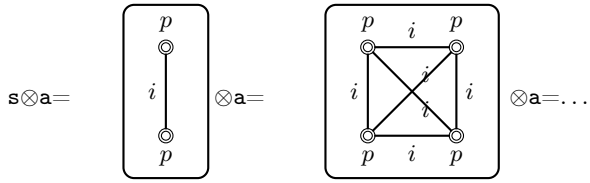


Figure 2.8: Repeated application of action a in the epistemic state s showing how product update can increase the size of an epistemic state.

2.4.5 State Reduction

If a world in an epistemic state fulfills the preconditions of more than one event in an action, then the product update can result in larger epistemic state in terms of the number of worlds in it.

Consider the epistemic state s and action a shown in figure 2.7. The state consists of a single world and the action of two event both having their precondition satisfied in the world. The resulting state from the product update between these are shown in figure 2.8 along with the result of reapplying the action in the resulting state.

Each resulting state captures the same knowledge as the initial state, e.g are bisimilar, but has grown two and four times in the number of worlds. A solution to the possible explosion in the number of worlds are proposed by Ditmarsch, Hoek, and Kooi [6] using the bisimulation contraction[11]. In this thesis states of often a result of a product update, and so the $\otimes_m$ is used to indicate the bisimulation contraction of the resulting state from the product update.

By taking the bisimulation contraction of the result of the product update between s and a, the result is again s.

[11] Reducing states to their minimal state modulo bisimulation

2.5 Discussion

When concerned with giving an autonomous agent the ability to reason about a course of action for achieving a goal, it is in many domains unrealistic to expect the agent to have a view corresponding to that of an external observer.

A planner for an autonomous agent should instead be able to model the state of affairs from the perspective of the agent. A way to achieve this with dynamic epistemic logic is by using local states as was described in section 2.4.2 on page 13.

When the agent during its reasoning process encounters two situation that it is unable to distinguish at the time of reasoning, but knows it will be able to distinguish at the time of execution, then it can reason about a different course of action for each situation. Such distinguishable states are named internal states, and one method for identifying these was proposed in section 2.4.2 on page 13.

This chapter has introduced some of the formalism of dynamic epistemic logic, and how it can be used as a basic for automated planning. Some of the formalism not included in this thesis are common knowledge, and having action modality as part of the epistemic language. The latter makes it possible to reason about actions in epistemic formulas. This allow goals formulas to, for example, require that a certain action should (not) be applicable. Including the action modality in the epistemic language is considered by both Andersen, Bolander, and Jensen [1], and Löwe, Pacuit and Witzel [2].

Common knowledge in a group of agents is knowledge that they share such that it is know by everyone, and everyone knows that everyone knows it, and everyone knows that everyone knows that everyone knows it ... and so on ad infinitum [6]. A lot of examples has been proposed, such as the muddy children puzzle, showing that when something is common knowledge among a group of agents, then it affects their line of reasoning. Common knowledge in multi-agent planning are treated by Andersen and Bolander [3].

Chapter 3

Plans and Planning Problems

The previous chapter described the complexity of the planning environment (partial-observable and non-deterministic). This chapter introduces the planning problem, the different solutions types for it, and the plans that represent such solutions. The main resources for this chapter are [1], [5], and [7].

The planning problem will not be treated as multi-agent in the classical sense where multiple agents are acting in the same environment. Instead it is considers an environment with multiple agents present, but only one acting agent (the planning agent).

3.1 Planning problem and domain

The following definition of a planning problem is the one defined by Andersen, Bolander, and Jensen [1], extended with the planning agent as a parameter. Explicitly stating the planning agent is necessary when dividing a state into its internal states of the planning agent.

Definition 3.1 (**Multi-Agent Epistemic Planning Problem [1]**). *Let $\mathcal{P}$ be a finite set of propositions, and let $\mathcal{A}$ a finite set of agents, then a planning problem on $(\mathcal{P}, \mathcal{A})$ is a triple $P = (s_0, i, E, \phi_g)$, where*

- *s_0 is an internal state on $\mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ representing the initial state from where the planning agent is trying to achieve the goal,*
- *$i \in \mathcal{A}$ is the planning agent,*
- *E is the set of actions available to the planning agent,*
- *ϕ_g is an epistemic formula on $\mathcal{L}_{\mathcal{K}}(\mathcal{P}, \mathcal{A})$ that should hold after a plan has been executed (the goal).*

The scenario from the prisoner and guard example can be describe by the planning problem P_{pg} with

$$P_{pg} = ((\mathcal{M}_0, \{w_1, w_2\}), p, \{\texttt{run}, \texttt{harass}, \texttt{bribe}\}, e \wedge (b \vee \neg K_g e)) \tag{3.1}$$

The epistemic planning domain is implicitly given through the planning problem. To reason about the reachable states in the state space of the domain these are explicitly defined through the function Γ below.

Definition 3.2 (**Multi-Agent Epistemic State Space**). *Let $P = (s_0, i, E, \phi_g)$ be a planning problem. The reachable states in the planning domain is defined recursively by $\Gamma(\{s_0\}, E, \gamma)$ where*

$$\Gamma(S, E) = S \cup \Gamma(\{s' \mid e \in E \wedge s \in S \wedge s' \in \gamma(s, a)\}, E)$$

$$\gamma(s, a) = \begin{cases} int_i(s \otimes a) & \text{if } a \text{ is applicable in } s \\ \{\} & \text{otherwise} \end{cases}$$

The state space contains only internal states of the planning agent, so that the internal perspective of the planning agent is kept throughout the reasoning process.

3.2 Solution types

A solution to a multi-agent epistemic planning problem is a plan that can be executed and possibly result in a state wherein the goal formula holds (from here on called a goal state). Ghallab, Nau, and Paolo [5] define three solution types to a planning problem in a partially-observable and non-deterministic environment.

- a *weak* solution has a chance to achieve a goal state,
- a *strong* solution guarantees to achieve a goal state,
- a *strong cyclic* solution guarantees to eventually achieve a goal state.

Execution of a plan that is a strong cyclic solution can repeatedly result in the same state (like flipping a coin and repeatedly having it land on head), but will under a *fairness* assumption eventually exit any loop (getting tails), resulting in a goal state.

The strength of the solutions are strong $>$ strong cyclic $>$ weak, and it follows that a strong solution is preferable over a strong cyclic solutions which in turn is preferable over a weak solution. As a consequence, there might exist a weak solution if none of the other exists and there might exists a strong cyclic solution when no strong solution exists. Indeed for the prisoner and the guard there does not exists a strong solution, but it will be shown later that both a strong cyclic solution and a weak solution exists. By relaxing the required strength of a solution, the number of possible solutions increases, and this is done in this thesis by considering strong cyclic and weak solutions.

State	Action
$\mathtt{s}_0$	`bribe`
$\mathtt{s}_3$	`run`

Table 3.1: A policy π_1 for the prisoner and guard shown as a table.

3.3 Conditional Plans

The following definition of a policy is by Ghallab, Nau, and Paolo [5], but is made partial and further restricted on the states it can contain, to fit the purpose of this thesis.

Definition 3.3 (Policies [5]). *Let $P = \{s_0, E, \phi_g\}$ be a planning problem, then a policy π for P is defined as a partial mapping of the states in the planning domain for P into actions*

$$\pi : S \to E, \text{ where } S \subseteq \Gamma(\{s_0\}, E, \gamma)$$

such that for any state-action pair (s, e) it holds that e is applicable in s, and there does not exist another state-action pairs (s', e') in π where $s \leftrightarrows s'$.

The lookup function $\pi(s)$ returns the action mapped to s if s is defined in π, and otherwise returns *undefined*.

Because bisimiliar states cannot be distinguished by the planning agent, they should never be mapped to different actions. It is incorporated into the policy definition by never having two bisimilar states in it. Instead when looking up a state s, the lookup function will return the action mapped to any state s' bismiliar to s.

Definition 3.4 (Policy Domain). *Given a policy π, the domain $D(\pi)$ is defined as the set of states in the state space for which the policy defines an action.*

$$D(\pi) = \{s \mid (s, e) \in \pi \text{ for some } e \in E\}.$$

In this thesis only finite plans will be considered, e.g all policy domains are finite.

Imagine the prisoner considers first to try and bribe the guard, and if the guard decides to accept the bribe, to run. The corresponding policy π_1 is shown in table 3.1, where $\mathtt{s}_0$ is the initial state and $\mathtt{s}_3$ is defined in figure 2.6 on page 16. The policy π_1 can be executed by an agent by sensing the current state ($\mathtt{s}_0$),

lookup and execute the corresponding action (`bribe`), sense the new state (either $\mathtt{s}_2$ or $\mathtt{s}_3$) and lookup and execute the corresponding action if it exists in π_1. The goal will only be achieved if the resulting state is $\mathtt{s}_3$, making π_1 a weak solution. In the following it is shown how it can be determined what kind of solution, if any, a policy represents.

3.3.1 Strong Cyclic and Weak Policies

Definition 3.5 (**Reachability in policies**). *Let π be a policy for a planning problem $P = (s_0, i, E, \phi_g)$, and $\mathtt{s}$ an internal state, then the set of states that can be reached from $\mathtt{s}$ using π is defined as $R^*_\pi(\mathtt{s})$, where*

$$R^j_\pi(s) = \begin{cases} R^0_\pi(s) = & \{s\} \\ R^{j+1}_\pi(s) = & \{s'' \mid s' \in R^i_\pi(s) \wedge s' \in D(\pi) \wedge s'' \in int_i(s' \otimes_m \pi(s'))\} \end{cases}$$

$$R^*_\pi(s) = \bigcup_{j \in \mathbb{N}} R^j_\pi(s)$$

The reachability of a set S' of internal states is defined as $R^*(S') = \bigcup_{s \in S'} R^j_\pi(s)$.

The possible reachable states from any state s using a policy π is therefore at most the states in the domain of π, along with any internal states from a product update between a state in $D(\pi)$ and its corresponding action. Because the latter produces a finite number of states and the domain of any policy is finite, so is the set of reachable state finite.

Definition 3.6 (**Proper Property**). *Let π be a policy for the planning problem $P = (s_0, i, E, \phi_g)$, and let s be a state in the state space $\Gamma(\{s_0\}, E)$, then π is proper with respect to s iff $\exists s' \in R^*_\pi(s) : s' \models \phi_g$.*

The *proper* property says that a policy is proper with respect to a state, if and only if a goal state is reachable from the state using that policy. The definition is extended to sets as a policy is proper with respect to a set of states if and only if it is proper with respect to each state in the set.

The notion of proper is from Bryce and Buffet [4], and used by Fu, Ng, Bastani, and Yen [7] to define strong cyclic policies, but their definition has been altered for the purpose of this thesis. In their work, they define a policy as proper with respect to a state s, if and only if $\forall s' \in R^*_\pi(s) \exists s'' \in R^*_\pi(s') : s'' \models \phi_g$. They use this as the definition of a strong cyclic solution to a planning problem. That their definition indeed describes a strong cyclic solution will be shown later in the chapter. The definition of a proper policy has been altered so as to fit both the strong cyclic and weak policy type, as they are introduced below.

Definition 3.7 (Policy Types). *Let $P = (s_0, i, E, \phi_g)$ be a planning problem, and let π be a policy on the planning domain of P. Then the policy for π is a*

Weak Policy	*iff π is proper with respect to s_0.*
Strong Cyclic Policy	*iff π is proper with respect to the set $R^*_\pi(s_0)$.*

Notice that the definition of strong cyclic is now the definition of a proper policy as it was introduced by Bryce and Buffet [4].

The definition of a weak policy is seen to capture the definition of a weak solution, and that a strong cyclic policy is a strong cyclic solution is shown in the following.

Lemma 3.1. *Let $P = (s_0, i, E, \phi_g)$ be a planning problem, and π a strong cyclic policy for it. Then π is a strong cyclic solution to P.*

Proof. Done by contradiction. Assume the existence of a strong cyclic policy π that is not a strong cyclic solution. Then, it must be possible to repeatedly use the policy from the initial state, and end in either

1. a non-goal state that is not defined by π, or
2. a loop that cannot be exited.

The former is a contradiction of definition 3.7 because it requires that all states reachable from the initial state can reach a goal state, and so must all reachable state that is not defined by the policy be goal states.

The later is a contradiction of definition 3.7 because the states in the never ending loop are reachable from the initial state, but it is not possible from any of them to reach a goal state. □

Strong cyclic (resp. weak) policy and strong cyclic (resp. weak) solution will be used interchangeably throughout the remaining of this thesis. The symbol $\alpha = sc|w$ will be used to denote a strong cyclic (resp. weak) solutions.

3.4 Discussion

3.4.1 Alternative Plan Representation

To represent conditional plans Andersen, Bolander, and Jensen [1] uses a *plan language*, containing the conditional construct **if** $K\phi$ **then** π **else** π, where π is a sentence in the plan language. The construct K (the agent is implicitly given because they consider the single-agent setting only) ensures that the planning agent can only make its choice of action based on states that are distinguishable to it. This approach requires the construction of a *distinguishable* formula that uniquely identifies a state for the conditional construct.

They further describe how a plan in the plan language can be converted into an epistemic formula allowing for plan verification through model checking in the initial state.

Policies was used in this thesis as an alternative plan representation, and the decision was based on the following.

1. Distinguishable formulas for multi-agent epistemic models are complex, and makes the resulting plans cumbersome.
2. The plan language can be extended to capture strong cyclic solutions, but how the translation to epistemic formulas for model checking should be defined are not obvious.

3.4.2 Strong Policy

The notion of proper is used to define both the strong cyclic and weak policy, indicating that these two types are related. And indeed both types are seen to be indifferent towards how a goal state is reached, but simply require a goal state to be reachable from every state (or the initial state in the weak case). On the contrary, the strong policy requires that goal states should be reachable without loops.

Strong plans are treated by Andersen, Bolander, and Jensen [1] and are therefore not considered in this thesis. This furthermore helps to keep the planning algorithm, proposed in the following chapter, more clear.

Chapter 4

Planning

This chapter introduces the planning algorithm for finding strong cyclic and weak solutions. The main resource for this chapter is [1]. The chapter first describes the planning graph used to search the state space, the expansion rule used to expand it (proposed by Andersen, Bolander, and Jensen [1] for planning trees), and then a novel approach for finding strong cyclic and weak solutions in the planning graph.

4.1 Plan Synthesis

The following two definitions describe the planning graph that represents the state space search, and the graph expansion rule that is used to search it (expand the graph). The planning graph is an AND/OR-graph, where OR-nodes represents branching made by the agent's choice and AND-nodes represent the branching made by the environment's choice of outcome for an action.

This idea is introduced by Ghallab, Nau, and Paolo [5], and used in the planning algorithm proposed by Andersen, Bolander, and Jensen [1]. In the latter they use a planning tree, but to find cyclic plans a planning graph is necessary.

Definition 4.1 (**Planning Graph [1]**). *A planning graph G is a labeled AND/OR-graph with directed edges, where each node n is labeled by an epistemic model $s(n)$, and each edge (n, m) leaving an OR-node is labeled by an action model $e(n, m)$.*

When creating a planning graph G for a planning problem $P = (s_0, i, E, \phi_g)$, the initial graph will consist of a single OR-node labeled with the initial state s_0, and this node will be denoted $init(G)$. The state space is hereafter explored by repeated application of the following graph expansion rule on the planning graph.

Definition 4.2 (**Graph Expansion Rule**). *Let G be a planning graph for a planning problem $P = (s_0, i, E, \phi_g)$. The graph expansion rule is defined as follows. Pick an node $n \in V_{OR}(G)$ and an action $e \in E$ applicable in $s(n)$ with the proviso that e does not label any existing outgoing edges from n. Then:*

1. *Add a new node m to $V_{AND}(G)$ with $s(m) = s(n) \otimes_m e$, and add an edge (n, m) with $e(n, m) = e$.*
2. *$\forall s' \in int_i(s(m))$ if $\exists n' \in V_{OR}(G)$ and $s(n') \leftrightarrows s'$, then add an edge (m, n'), else add an node m' with $s(m') = s'$ to $V_{OR}(G)$ and an edge (m, m').*

A node is defined as *fully expanded* if all the applicable actions for the labeled state are labeled on outgoing edges from the node. When applying the graph expansion rule, the selection of nodes are restricted in the following manner.

$\mathcal{S}$. The graph expansion rule may not be applied to a node if there exists another node of lesser depth with respect to $init(G)$, which has not yet been fully expanded.

With this restriction, repeated application of the expansion rule corresponds to a breadth-first expansion, since no node will ever be expanded before all nodes of lesser depth have been fully expanded.

The purpose of the state space search is to find goal nodes, and once a goal node is found, it should therefore not be expanded. To ensure that the graph expansion rule is not applied to goal nodes, the following blocking condition is used.

$\mathcal{B}$. The graph expansion rule may not be applied to a node n if $s(n) \models \phi_g$ [1].

In the remaining of the thesis when the graph expansion rule is mentioned, it will implicitly be with the restriction $\mathcal{S}$ and blocking condition $\mathcal{B}$, unless otherwise stated.

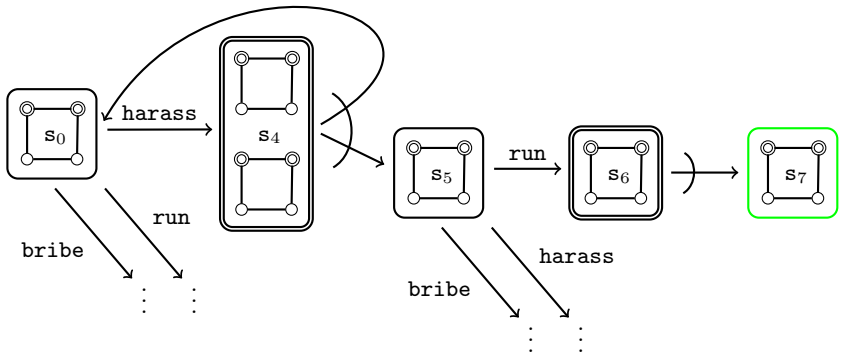
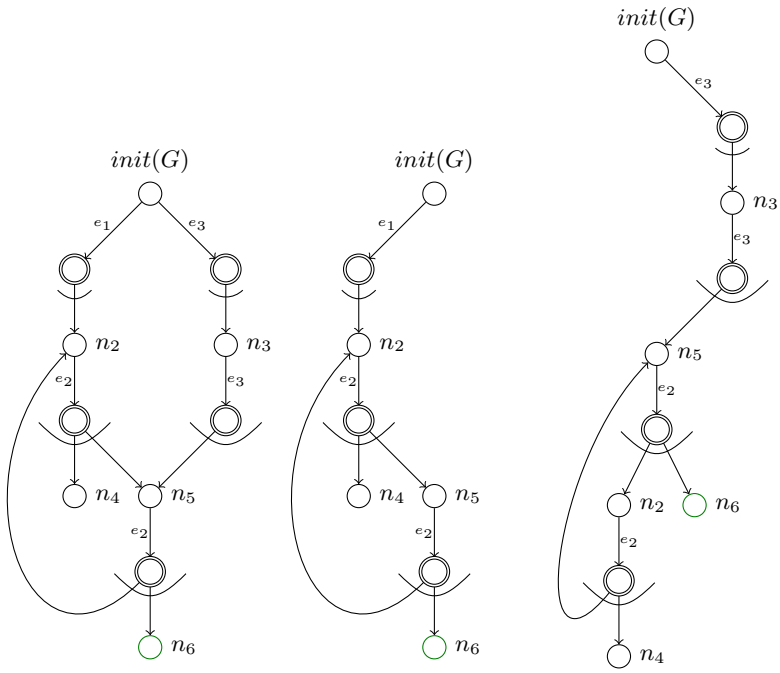


Figure 4.1: A section of the planning graph G for the planning problem P_{pg} (3.1) for the prisoner and the guard example. The AND-nodes are shown with double lines, the states labeled on nodes are shown within with labels removed for simplicity, and goal nodes are marked with green.

Imagine that the prisoner considers a course of action where he keeps harassing the guard until the guard is no longer facing the exit, and then make a run for it. The part of the planning graph exploring this strategy is shown in figure 4.1. The AND-nodes are shown with double circles, and all nodes are shown with (a simplified version of) their labeled epistemic states. The result of applying the graph expansion rule to the initial node with the action `harass` is an AND-node with its labeled state consisting of two internal states. One of the internal states is the initial state, resulting in a back edge to the initial node. The second internal state is labeled on a new node to which the graph expansion rule is applied. When the action `run` is applied, it results in the OR-node labeled with state $\mathtt{s}_7$ which is a goal state making the node a goal node. Due to blocking condition $\mathcal{B}$, the node with $\mathtt{s}_7$ will not be further expanded.

The purpose of the state space search is to eventually map states to actions and create a plan for the planning problem. This means that for each OR-node only one of the actions labeled on the outgoing edges can be selected for the policy. For this, the following notion of a *policy graph* is introduced.



(a) G *(b) H_1* *(c) H_2*

Figure 4.2: (a) shows a planning graph G, and (b) and (c) shows the two policy graphs H_1 and H_2 in it. Node n_4 is yet to be expanded, and node n_6 is a goal node.

Definition 4.3 (Policy Graph). *Let G be a planning graph, then a policy graph H is any partial subgraph of G where the following holds:*

- $init(G) \in H$
- $\forall n \in V_{OR}(H)$ *: n has at most one outgoing edge*
- $\forall n \in V_{AND}(H)$ *: n has the same number of outgoing edges as in G*
- *Leaf nodes in H are leaf nodes in G*

The figure 4.2 illustrates how a not yet saturated planning graph G is divided into two policy graphs, and how each of these corresponds to looking at the different policies that can be created by following the paths in the planning graph. In figure 4.2c the action e_3 was chosen from the starting position, and in figure 4.2b the action e_1 was chosen corresponding to the two possible choices that the planning graph has encountered so far. A policy graph is said to represent a policy, and if it represents an α-policy, then the policy graph is

called α-solved [1].

Definition 4.4 (**Goal Node Reachability**). *Let $P = (s_0, i, E, \phi_g)$ be a planning problem, let G be a planning graph build for P, and let H be a policy graph in G. We then define the set of nodes $N(H)$ in H that can reach an goal nodes by*

$$n \in N(H) \text{ iff } n \in V_{OR}(H) \text{ and } \exists m \in V_{OR}(H) : n \rightarrow_H^* m \wedge s(m) \models \phi_g.$$

Definition 4.5 (**Solved Policy Graph**). *Let $\alpha = sc|w$, let G be a planning graph for a planning problem P, and let H be a policy graph in G. H is then called α-solved if the following holds*

- *if $\alpha = w$, then $|N(H)| \geq 1$.*
- *if $\alpha = sc$, then $V_{OR}(H)$=$N(H)$.*

The two policy graphs shown in figure 4.2 on the facing page are both α-solved if $\alpha = w$, because there exists one goal node n_6 in them. The resulting plan would be optimistic (e.g. weak) because it depends on the environment selecting n_5 over n_2 to reach the goal node. However, if $\alpha = sc$, then neither of the two policy graphs are α-solved because the node n_4 cannot reach a goal node, making $V_{OR}(H) \neq N(H)$.

Given a policy graph, a policy is constructed directly from it by mapping the states labeled at OR-nodes in the policy graph with the action labeled on their outgoing edge.

Definition 4.6 (**Policies for Policy Graphs**). *Let H be a policy graph, then the policy for H is defined by*

$$\omega(H) = \{(s(n), e(n, m)) \mid n \in V_{OR}(H) \wedge \exists (n, m) \in E(H)\}$$

The following lemma ensures that there are not added bisimilar states mapped to different actions in the policy. This ensures that the policy created from a policy graph is as defined in definition 3.3 on page 24.

Lemma 4.1. *When creating the policy for an execution graph, there will never be two bisimilar states added to the policy, mapped to different actions.*

Proof. When the planning graph is build, the second step in the graph expansion rule ensures that there will never exist two OR-nodes in the graph with bisimilar

[1] Recall $\alpha = sc|w$, where sc is the strong cyclic solution and w is the weak solution as introduced in section 3.2 on page 23.

labeled states. A policy is created from a policy graph, where the state of each non-leaf OR-node is labeled with the action labeled on the outgoing edge from that node. Because a policy graph per definition only has one outgoing edge from any OR-node, it is not possible to map the same state to different actions. □

With the above, a parameterized planning algorithm can now be defined for finding an α-solution to a planning problem P depending on which α is given as input.

Algorithm 1 **PLAN**(α,P)

```
1: let G be a planning graph consisting only of the OR-node n labeled by the
   initial state of P.
2: while G can be expanded do
3:     expand G using the graph expansion rule.
4:     if there exists an α-solved policy graph H in G then
5:         return ω(H).
6:     end if
7: end while
8: return FAIL
```

PLAN(α,P) will, if no α-solved policy graph is found, continue to expand the graph until every state in the planning domain has been labeled on a node in it. Because the algorithm perform a breadth-first search, it will if a solution exists eventually find it. The claim will be proven in the following chapter. However, Andersen and Bolander [3] [Theorem 19] states that with three or more agents, the plan existing problem for a multi-agent epistemic planning problem is only semi-decidable. It can therefore not be guaranteed that **PLAN**(α,P) will terminate when given a planning problem for which no solution exist.

4.2 Discussion

4.2.1 Optimal Plan

When termination cannot be guaranteed it introduces problems in terms of finding an optimal plan. With unordered actions, i.e. actions with no difference in cost, an optimal linear plan is defined as the shortest plan in number of distinct states visited before a goal state is reached. When comparing optimality of two conditional plans it is by largest number of distinct states visited on the longest path from the initial state to a goal state. This corresponds to a worst case scenario when executing the plan. An optimal conditional plan is one where there does not exist another plan with a shorter worst case.

If the expansion of a node results in multiple solved policy graphs, then it is not defined how the planning algorithm should chose between these. Without such reasoning, the returned policy cannot be guaranteed to be optimal. If assuming that the planning algorithm is somehow able to determine which of these policy graph will result in the most optimal plan, then the planning algorithm can be guaranteed to return an optimal weak plan, due to the breadth-first search.

However, as illustrated in figure 4.3 on the following page, for a strong cyclic plan this is not the case. In the figure, the node n_1 has just been expanded resulting in a back edge. This creates a strong cyclic solved policy graph, with the choice e_2 at the initial node. The length of the policy graph in terms of OR-nodes are 5 (same as the worst case for the resulting plan). But, illustrated with the dotted lines, if the node n_2 had been expanded in the next iteration then it would have created another goal node, and made the second policy graph strong cyclic solved as well. As this has a longest path of 4 nodes, so would this be a more optimal solution.

The planning algorithm can as a consequence not be guaranteed to return an optimal solution.

4.2.2 Tractability

The number of states in the state space can explode when increasing the number of available action. Moreover are the states them self able to grow, as they are a result of a previous state applied with an action. It is noted by Andersen and Bolander [3], that there in the general case are no upper bound in the size of states reachable in the state space, even when considering only the bisimulation

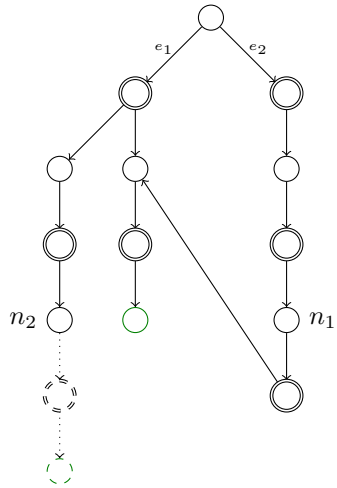


Figure 4.3: A planning graph containing two policy graphs, one for the choice e_1, and one for the choice e_2. The node n_1 has just been expanded resulting in the back edge, and an α-solved policy graph. The dotted nodes and edges are what will be, if the node n_2 were expanded next.

contraction of states. How to reduce the state space is still an open question, but Löwe, Pacuit and Witzel [2] proposes a solution, and shows how it can keep the state space of a concrete scenario small. They do so by imposing restrictions on the type of actions to be used. Their idea is that repeated application of an action must only produce bisimilar states, and that if a sequence of actions are applied to a state, then the resulting state should be the same indifferently of the order of the sequence. How to generalize their approach and identify such actions is still unclear.

This thesis is not concerned with the computational complexity of the proposed algorithm, but notes that the plan existence problem for strong plans in a partial-observable non-deterministic environment is shown to be in the 2EXP-time class [8], and is therefor in general a very hard problem.

Chapter 5

Soundness and Completeness

The planning algorithm proposed by Andersen, Bolander, and Jensen [1] is shown both sound and complete. In the following, these properties are proven for the planning algorithm proposed in this thesis.

5.1 Proofs

The following lemma describes the relation between the states in a policy graph and the states reachable through the policy created from it.

Lemma 5.1. *Let H be a policy graph and let n be an node in $V_{OR}(H)$ then the set of reachable states from $s(n)$ using the policy $\omega(H)$ is exactly the set of states labeled on nodes in $V_{OR}(H_n)$.*

Proof. This corresponds to prove that $R^*_{\omega(H)}(s(n)) = s(V_{OR}(H_n))$[1]. The proof is done by induction on the height h of OR-node n in H.

[1] Given a set of nodes the set of states labeled the nodes are defined in the standard way as $s(V) = \{s(n) \mid n \in V\}$.

Base case is $h = 0$, then $s(V_{OR}(H_n)) = \{s(n)\}$. This means that n is a leaf node, and definition 4.6 on page 33 then says that $s(n)$ is not defined in $\omega(H)$. Then $R^*_{\omega(H)}(s(n)) = \{s(n)\}$, concluding the base case.

For the inductive step, assume that for any node n with height $l \leq h$, it holds that $R^*_{\omega(H)}(s(n)) = \{s(n') \mid n' \in V_{OR}(H_n)\}$. It will now be proven that for any OR-node n'' of height $h+2$ such that $n'' \to n' \to n$ it holds that $R^*_{\omega(H)}(s(n''))$ $= s(V_{OR}(H_{n''}))$.

From definition 4.2 on page 30 n' will be an AND-node with children $\{m_1, ..., m_k\}$ such that each child is labeled with a state in $int_i(n'' \otimes_m e(n'', n'))$. The set of OR-nodes in the subgraph $H_{n''}$ can be described as the union of the OR-nodes in the subgraphs of each of its child nodes, and n'' itself.

$$V_{OR}(H_{n''}) = \{n''\} \cup V_{OR}(H_{m_1}) \cup ... \cup V_{OR}(H_{m_k}) \tag{5.1}$$

And the states labeled on the nodes in this set can be described as

$$\{s(n'')\} \cup s(V_{OR}(H_{m_1})) \cup ... \cup s(V_{OR}(H_{m_k})) \tag{5.2}$$

Let $\{s_1, ..., s_j\} = int_i(s(n'') \otimes_m \omega(H)(s(n'')))$, then from definition 3.5 on page 25 it is given that

$$R^*_{\omega(H)}(s(n'')) = \{s(n)\} \cup R^*_{\omega(H)}(s_1), ..., R^*_{\omega(H)}(s_j) \tag{5.3}$$

From definition 4.6 on page 33 we know that $\omega(H)(s(n'')) = e(n'', n')$, and we can therefore expand equation (5.3) as follows.

$$R^*_{\omega(H)}(s(n'')) = \{s(n)\} \cup R^*_{\omega(H)}(s(m_1)) \cup ... \cup R^*_{\omega(H)}(s(m_k)) \tag{5.4}$$

The inductive hypothesis says that for each child m in $\{m_1, ..., m_k\}$ it holds that $R^*_{\omega(H)}(s(m)) = s(V_{OR}(H_m))$. Therefore (5.2) and (5.4) described the same set of states, which concludes the inductive step. □

Theorem 5.2 (Soundness). *Let $\alpha = sc/w$, and let H be an α-solved policy graph found by the planning algorithm for a planning problem $P = \{s_0, i, E, \phi_g\}$. Then the policy $\omega(H)$ is an α-solution to P.*

Proof. Case $\alpha = sc$. The definition for policy types, definition 3.7 on page 26, states that the following must hold for the policy $\omega(H)$ to be a strong cyclic policy.

$$\forall s \in R^*_{\omega(H)}(s_0) \exists s' \in R^*_{\omega(H)}(s) : s \models \phi_g \tag{5.5}$$

Because H is α-solved so must the definition of an α-solved policy graph (definition 4.5 on page 33) hold for H. The definition uses the set $N(H)$ to determine

if H is α-solved. Using the definition for goal node reachability (definition 4.4 on page 33), the nodes in this set $N(H)$ can be described as

$$\forall n \in N(H) \exists m \in V_{OR}(H) : (n \rightarrow^* m) \wedge s(m) \models \phi_g \tag{5.6}$$

Since H is α-solved, then $N(H) = V_{OR}$ and (5.6) can be written as

$$\forall n \in V_{OR}(H) \exists m \in V_{OR}(H) : (n \rightarrow^* m) \wedge s(m) \models \phi_g \tag{5.7}$$

If there exists a path from some node n to a node m in H, then m is also part of the subgraph H_n of H, and (5.7) can be written as

$$\forall n \in V_{OR}(H) \exists m \in V_{OR}(H_n) : s(m) \models \phi_g \tag{5.8}$$

From lemma 5.1 on page 37 it is given that the reachable states $R^*_{\omega(H)}(s_0)$ are exactly the set of states labeled on nodes in $V_{OR}(H_{init(G)})$ e.g. in $V_{OR}(H)$.

Furthermore it is given that for any node $n \in V_{OR}(H)$ the set of states $R^*_{\omega(H)}(s(n))$ is exactly the set of states labeled on nodes in $V_{OR}(H_n)$.

This means that (5.8) can be rewritten as

$$\forall s \in R^*_{\omega(H)}(s_0) \exists s' \in R^*_{\omega(H)}(s) : s' \models \phi_g \tag{5.9}$$

Since (5.9) is precisely the definition of a strong cyclic policy, it concludes the proof for the policy $\omega(H)$ being a strong cyclic solution to P.

Case $\alpha = w$. For the policy $\omega(H)$ to be a weak policy the following must hold.

$$\exists s \in R^*_{\omega(H)}(s_0) : s \models \phi_g \tag{5.10}$$

lemma 5.1 on page 37 states that $s(V_{OR}(H)) = R^*_{\omega(H)}(s_0)$. Thus, if it can be proven that

$$\exists s \in s(H) : s \models \phi_g$$

holds, then so does (5.10). Because H is α-solved then $|N(H)| > 0$, i.e. there exists a node n in $V_{OR}(H)$ with $s(n) \models \phi_g$. This concludes the proof of the policy $\omega(H)$ being a weak solution for P. □

Theorem 5.3 (Completeness). *Let $\alpha = sc/w$. If there exists a policy that is an α-solution to the planning problem $P = (s_0, i, E, \phi_g)$, then the plan algorithm will find a policy graph that is α-solved.*

Proof. The definition 4.6 on page 33 describes how a policy graph can be turned into a policy. Let H be a policy graph that would result in the policy π, and let π be an α-solution to the planning problem P. The proof will now be divided

into two cases: the case where the planning graph contains another α-solved policy graph H' before it contains H, and the case where it does not.

Case 1. There can exist more than one solution to a planning problem, but the planning algorithm will always stop when it encounters the first α-solved policy graph. If the planning algorithm constructs an α-solved policy graph H' before H is constructed, then this will be found because the planning algorithm considers all policy graphs in the planning graph.

Case 2. If there does not exist an α-solved policy graph H', that is found by the planning algorithm before H, then G will always find H. The proof will be divided into four parts, first part will show how an policy π can be represented as a digraph Σ, second part will show how such a policy representation can be converted to a policy graph, third part will prove that such an policy graph will be α-solved if the policy is an α-solution, and fourth part will prove that if such policy graph exists, then it will be contained in then planning graph G for the planning problem P.

Part 1. Let π be an α-solved policy for the planning problem $P = (s_0, i, E, \phi_g)$, then π can be represented as a digraph $\Sigma = (V, E)$, where $V = R^*_\pi(s_0)$ is the nodes in the graph, and $E = \{(s, s') \mid s \in R^*_\pi(s_0) \wedge s \in D(\pi) \wedge s' \in int_i(s \otimes_m \pi(s))\}$ is the edges.

The reachable nodes from any node s in Σ can now be described by

$$\begin{aligned}
\{s' \mid s \rightarrow^0 s'\} &= \{s\} &&= R^0_\pi(s)\\
\{s' \mid s \rightarrow^1 s'\} &= \{s'' \mid s' \in \{s' \mid s \rightarrow^0 s'\} \wedge s' \in D(\pi) \wedge s'' \in int_i(s' \otimes_m \pi(s'))\} &&= R^1_\pi(s)\\
\{s' \mid s \rightarrow^{i+1} s'\} &= \{s'' \mid s' \in \{s' \mid s \rightarrow^{i-1} s'\} \wedge s' \in D(\pi) \wedge s'' \in int_i(s' \otimes_m \pi(s'))\} &&= R^{i+1}_\pi(s)
\end{aligned}$$

This leads to $\forall s \in V(\Sigma) : \{s' \mid s \rightarrow^* s'\} = R^*_\pi(s)$

Part 2. From the graph representation Σ of the policy π, a policy graph H is created by doing the following for all $s \in \Sigma$:

1. create an OR-node n with $s(n) = s$.
2. if s is a non-leaf node then
   (a) create an AND-node m labeled with $s(n) \otimes_m \pi(s(n))$
   (b) for all edges (s, s') in Σ make an edge (m, m') in H where $s(m) = s'$.
   (c) create the edge (n, m) with $e(n, m) = \pi(s)$.

Then H is an AND/OR-graph with alternating AND/OR-nodes, where $s(init(H)) = s_0$, each none-leaf OR-node has exactly one outgoing edge labeled with an ac-

tion, and each AND-node has a child node for each internal state in the state labeled on it.

From 1. the following then holds

$$s(V_{OR}(H)) = V(\Sigma) = R^*_\pi(s_0) \tag{5.11}$$

And 2. ensures that the paths in Σ are preserved in H, whereby it holds that

$$\forall n \in V_{OR}(H) : s(\{n' \mid n \rightarrow^* n' \wedge n' \in V_{OR}(H)\}) = R^*_\pi(s(n)) \tag{5.12}$$

Part 3. It will now be shown for each case of $\alpha = sc|w$ that if π is an α-solution, then H will be α-solved.

For $\alpha = sc$, it must be shown that $V_{OR}(H) = N(H)$, which by definition 4.4 on page 33 means

$$\forall n \in V_{OR}(H) : n \rightarrow^* m \wedge s(m) \models \phi_g \tag{5.13}$$

The policy π is strong cyclic, and the following requirement for strong cyclic must therefore hold. $\forall s \in R^*_\pi(s_0) \exists s' \in R^*_\pi(s) : s' \models \phi_g \overset{(5.11)}{\Leftrightarrow} \forall n \in V_{OR}(H) \exists s' \in R^*_\pi(s(n)) : s' \models \phi_g \overset{(5.12)}{\Leftrightarrow} \forall n \in V_{OR}(H) \exists n'' \in \{n' \mid n \rightarrow^*_H n' \wedge n' \in V_{OR}(H)\} : s(n'') \models \phi_g \Leftrightarrow \forall n \in V_{OR}(H) \exists n' \in V_{OR} : n \rightarrow^*_H n' \wedge s(n') \models \phi_g$. This concludes the proof that H is α-solved.

For $\alpha = w$ it must be shown that $|N(H)| > 0$, e.g there must exist a goal node in H. Because π is an α-solution, then it holds that π is proper with respect to s_0. The definition of proper is given as $\exists s \in R^*_\pi(s_0) : s \models \phi_g$. From (5.11) it holds that $s(V_{OR}(H)) = R^*_\pi(s_0)$, and so there is a node n in H where $s(n) \models \phi$, and n is therefor a goal node. This concludes the proof that H is α-solved.

Part 4. What is left to show is that H will be (part of) a policy graph in the planning graph G constructed for the same planning problem P using the graph expansion rule.

The initial node for both graphs is labeled with the same state s_0, and every action labeled on the outgoing edge from any OR-node in H is per definition applicable in the state labeled on the OR-node.

Now let n be any OR-node in H, where there exist an OR-node n' in G, with $s(n) \leftrightarrows s(n)'$. When the graph expansion rule is applied to n' with the action $e(n, m)$ labeled on the outgoing edge (n, m), then it will create the AND-node m', where $s(m') \leftrightarrows s(m)$. As a consequence m and m' will have child OR-nodes labeled with bisimilar states.

Because the planning algorithm performs a breadth-first search, so will all states labeled on node of dept k in H with respect to $init(H)$, be labeled on nodes in G, when G has been fully expanded at dept k.

Thus when G has been fully expanded at height equal to the height of H, then H will be contained in a policy graph in G, and this will then be α-solved.

This concludes the proof for completeness.

□

5.2 Discussion

A planning problem can have multiple solutions, but the planning algorithm cannot be guaranteed to return a specific solution because it always returns the first solution found. The consequence is as noted above that the proof for completeness is dependent on theorem 5.2 on page 38 showing soundness, to ensure that any solution returned, that is not the one assumed to exist, is also an valid solution.

Chapter 6

Implementation

This chapter briefly describes the implementation of **Plan**(α,P). The implementation is largely build upon the implementation of dynamic epistemic logic by Eijck [13]. The motivation for the implementation is to demonstrate the proposed planning algorithm. As it is one of the only planners with its expressiveness it also provides a base for further work, or possibly for an implementation in a (small) practical setting.

The implementation of the planning algorithm is used in a web application, chosen because it allows for easy access and use. Through the application the user can define an epistemic planning problem, and the resulting solved policy graph will be displayed, if one exists.

6.1 Related work

In the later years two practical implementations of the DEL framework have been developed.

DEMO by Eijck [13] standing for Dynamic Epistemic MOdeling tool, is an implementation of DEL written in the functional programming language Haskell.

DEMO can model epistemic updates, graphical display update results and, evaluate formulas in epistemic models.

moDELchecker by Witzel [14] is an implementation of DEL in the programming language C++, and includes a planning algorithm for synthesis of simple plans, i.e. a linearly ordered sequence of actions. The planner plans in the multi-agent setting from the perspective of an external observer. Both implementations show the usefulness of dynamic epistemic logic, not only as a theoretical framework, but in a practical setting as well.

DEMO has been chosen as the foundation for the implementation of the fragments of DEL used by this thesis because; it provides a straightforward implementation of DEL in the sense that it is easy to see how the implementation corresponds to the theory, it includes most of the elements used in this thesis, and it is the best documented.

6.2 Choice of technology

The implementation in this thesis uses the functional programming language F#, the object-oriented programming language C#, and the *Graphviz* engine [10] for generating graphs for the visualization. The elements from DEMO is ported to F#, and the planning algorithm is implemented in C#. The choice of technologies is made because of the familiarity with them, as well as the ease with which they integrate.

6.3 System Description

The main components and the flow in the system is shown in figure 6.1 on the next page, where only the web page rendering and input /output handling it left out.

6.3.1 DEL Library

The fragments ported from the implementation in DEMO are; *epistemic states*, *actions*, *product update*, *formula evaluation*, and *bisimulation contraction*. The implementation is except for naming convention, and minor language differences between F# and Haskell, kept as similar to that of DEMO as possible. The

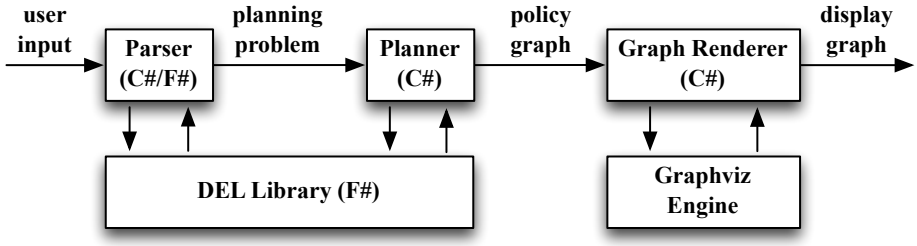


Figure 6.1: Program flow. Web page handling are left out.

representation of epistemic states and actions are shown in figure 8.1 on page 55 in appendix.

The ported implementations will not be described further as they are well documented [13]. The bisimulation contraction in DEMO was done by Sietsma [12], and is an implementation based on the three partition refinement algorithm proposed by Paige and Tarjan [9]. The added features to the DEL component is the division of a states into internal states, and the check for the existence of an bisimulation relation between states.

6.3.2 Planner

Planning Graph. The planning graph is represented by a list *Nodes* containing all nodes in the graph, and a queue *NodeQueue* containing the OR-nodes that has not yet been expanded e.g. are leaf nodes. Each node contain the epistemic state labeled on it, as well as a list of ingoing edges, and a list of outgoing edges. Edges contain a reference to the nodes connected by it and the action that is labeled on it, if any.

Planning Algorithm. The planning algorithm initializes the planning graph with a single node containing the initial state. It then keep expanding the graph one node at a time until either a solved policy graph is found, or no unexpanded nodes exists, or the application runs out of memory. The next node to be expanded is always the first on the queue, and any newly created OR-nodes are added to the queue, thereby implementing the breadth-first search. When expanding a node, all of the available actions are tried before expanding another node. Each expansion of a node is described in algorithm 3 on page 56 in appendix, and it is a directly implementation of the graph expansion rule from section 4.1 on page 29.

6.3.3 Policy Graph Handling

Policy Graph. A policy graph is implemented as an object containing a list of references for the nodes and edges in it, along with a reference to the initial node.

Policy Graph Update. When a node is expanded then one or both of the following happens.

1. Existing policy graphs containing the expanded node are updated to ensure they keep with the requirements for a policy graph in definition 4.3 on page 32.
2. If the expansion created a branch on an OR-node then a new policy graph is created.

Six cases are considered when updating the policy graphs after the expansion of a node. These are illustrated in figure 6.2 on page 48 and figure 6.3 on page 48, where the blue node has been expanded using the action $e1$, and the red edges and nodes are the ones created in the expansion.

Case 1-4 each requires an update of existing policy graphs whereas the two cases 5 and 6 require that new policy graphs are created. The main algorithm for maintaining policy graphs are outline in algorithm 2 on page 49, where the input are

- n the expanded node,
- n_{AND} the resulting AND-node,
- e_{AND} their connecting edge,
- N_{OR} the list of newly created child OR-nodes of n_{AND},
- E_{OR} the list of edges connecting n_{AND} with all of its child nodes.

Line 1-7 finds the existing policy graphs, or create new ones, that should be updated. Line 8-23 updates the policy graphs with the newly created nodes and edges. Line 24 stores the updated policy graphs.

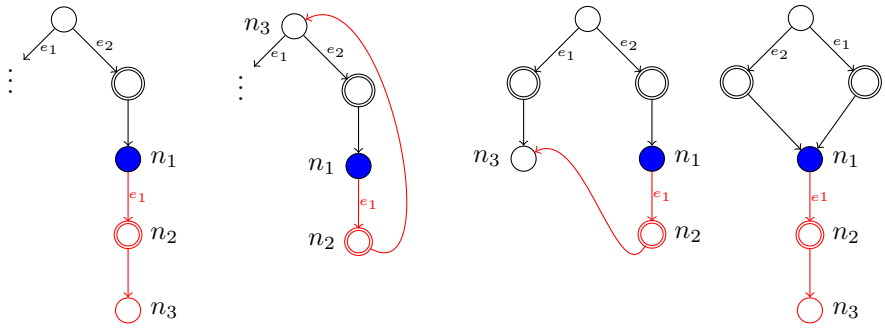
The two subroutines `ComputeUp` and `ComputeDown` creates new policy graphs. The former handles case 5 where a new branch is created on the expanded OR-node, and all the different policy graphs leading from the initial node to the expanded node should be copied for the new branch.

The result of `ComputeUp` in case 5 is a policy graph leading from the initial node to n_1. The created nodes and edges are then added to it during the update. This ensures that the existing policy graph, taking the action e_2 at n_1, is preserved.

The later handles case 6, where an edge is created between n_2 and an existing non-leaf OR-node n_3 in the graph. The subroutine take the existing OR-node, and a policy graph *policy* to be updated, as parameters. The subroutine then finds all the different policy graphs possible when considering n_3 as the initial node. For each of these policy graphs are the *policy* copied and extended with it. The subroutine returns a list of extended copies, and the original *policy* is replaced with these.

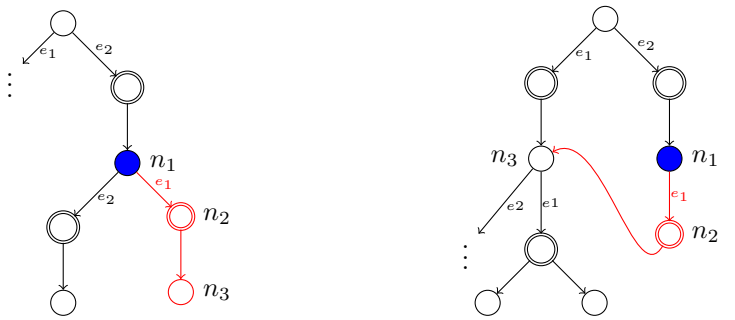
The result of `ComputeDown` in case 6 is two policy graphs, both consisting of the nodes $\{init(G), n_1, n_2, n_3\}$, but one containing the nodes reachable from n_3 using e_1, and the other the nodes reachable using e_2.

After the expansion of a node, when the affected policy graphs have been updated, each of them are check for being solved.



(a) Case 1. *(b)* Case 2. *(c)* Case 3. *(d)* Case 4.

Figure 6.2: The different results from expanding a node in the planning graph that only requires existing policy graphs to be updated. The blue node is expanded, and the red nodes and edges are created in the expansion.



(a) Case 5. *(b)* Case 6.

Figure 6.3: The different results from expanding a node in the planning graph that requires that new policy graphs are created. The blue node is expanded, and the red nodes and edges are created in the expansion.

Algorithm 2 The algorithm for update the list of policy graphs after an node has been expanded in the planning graph : `Update`$(n, N_{OR}, E_{OR}, n_{AND}, e_{AND})$

```
1: let P_all be the list of all policy graphs before expansion
2: if number of outgoing edges from n is greater than 1 then
3:     let P_ex ← ComputeUp(n)                                  ▷ Case 5.
4: else
5:     let P_ex be the list of policies that n is part of.      ▷ Case 4.
6:     remove P_ex from P_all
7: end if
8: let P_update be an empty list
9: for each policy in P_ex do
10:    add policy to P_update
11:    add n_AND, e_AND, E_OR, and N_OR to policy               ▷ Case 1.
12:    for each (n_and, m) in E_OR do
13:        if m is not in policy then                           ▷ Case 2.
14:            if m has no outgoing edges then                  ▷ Case 3.
15:                add m to policy
16:            else
17:                let P_copy ← ComputeDown(m, policy)          ▷ Case 6.
18:                remove policy from P_update
19:                add P_copy to P_update
20:            end if
21:        end if
22:    end for<
23: end for
24: add P_update to P_all
```

6.3.4 Screenshots

Graphical User Interface. The screenshot in figure 6.4 shows the graphical user interface, consisting of input fields for the solution type and planning problem. The input for the planning problem consists of inputs for the initial state, the planning agent, the goal formula, and the available actions.

Due to the possible large size of the output, the user can chose between having the output graph shown without labeled states, with labeled states on OR-nodes, or with labeled states on all nodes.

When the application returns it will display one of the messages; *solution found*, *no solution exists*, or *ran out of memory*. In the first case the output graph will show the solved policy graph, and in the two latter it will display the entire planning graph.

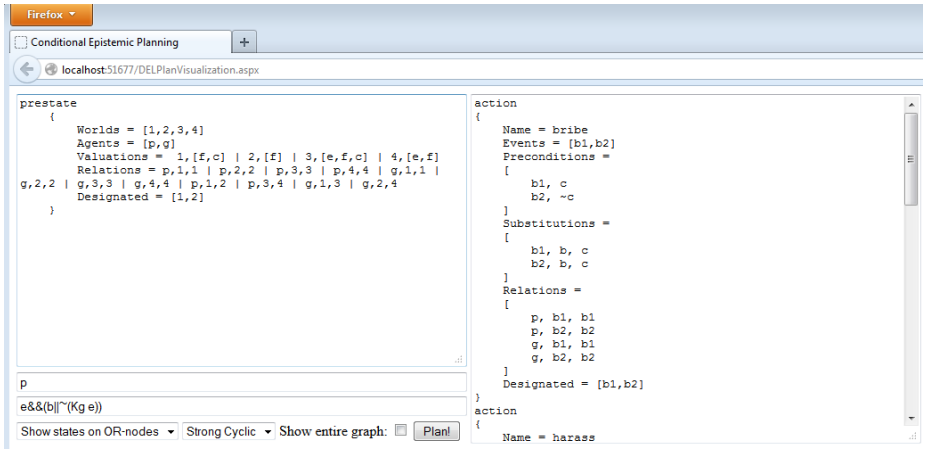


Figure 6.4: User interface. Input of the planning problem for the prisoner and guard example, in the browser.

Output Graph The screenshot in figure 8.3 on page 57 shows the graph presented to the user for the prisoner and guard example, when searching for a strong cyclic solution. The outgoing edges from AND-nodes are dotted, the goal nodes are marked with green, and the epistemic states labeled on nodes are shown as subgraphs. The reflexive loop are left out in the epistemic states, only the propositions that are true are shown at each world, and the actual worlds are colored blue.

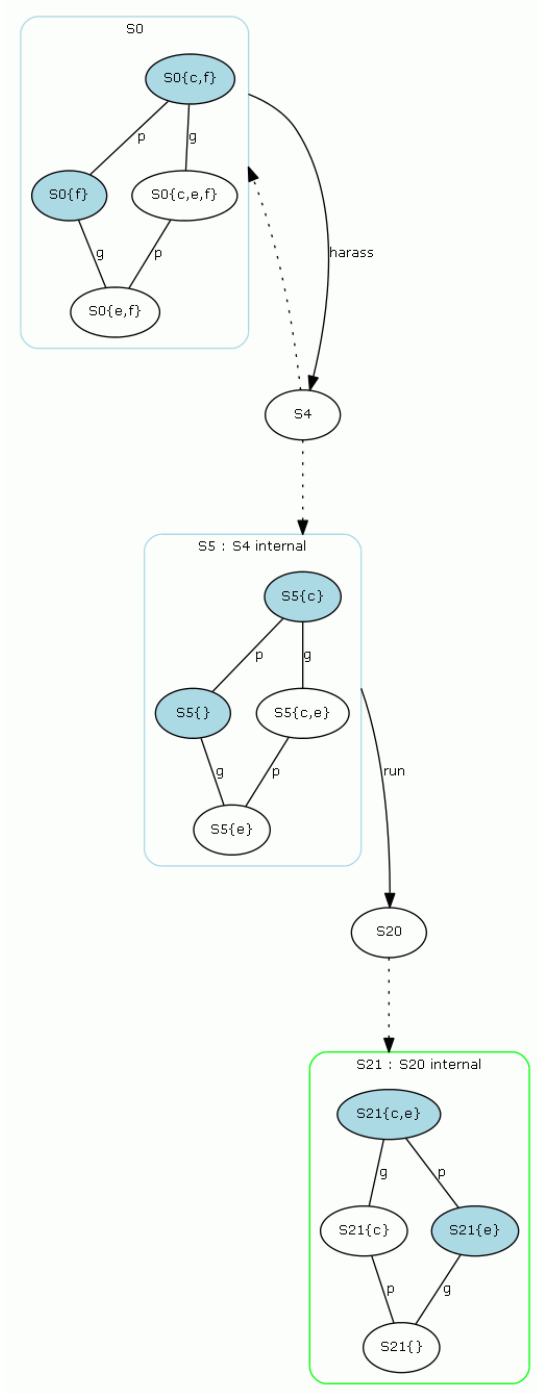


Figure 6.5: Output from the prisoner and guard example with $\alpha = sc$.

6.4 Discussion

6.4.1 Benchmarks

Time has not permitted in depth testing nor benchmarking of the implementation. A natural next step would be to try and compare it against existing planners for partial-observability. It is expected that it will not fair well, since the focus in this thesis has been on expressiveness, and not on the complexity of the computations in terms of speed or space. Another approach is to try and formulate a set of benchmarks for this type of planning with higher-order knowledge since this does not appear to have been done previously in the literature.

6.4.2 Policy Graph Handling

How policy graphs are found is left open in the definition of the planning algorithm in chapter 4 on page 29. The correctness of the approach used in the implementation has only been described informally, and it has only been tested using the prisoner and the guard example. Running the example with a goal formula that cannot be fulfilled ($\bot$) results in a saturated graph with a total of 59 OR-nodes and 293 policy graphs, which is as expected. The resulting planning graph without the epistemic states shown, can be found in figure 8.3 on page 57 in appendix.

Chapter 7

Conclusion

A planning algorithm **Plan**(α,P) has been proposed based on dynamic epistemic logic that allows for modeling of higher-order knowledge. The algorithm finds strong cyclic and weak plans in a multi-agent scenario with applicability, incomplete knowledge in the the initial state, and non-deterministic action. The algorithm uses the perspective of the acting agent in the environment instead of that of an external observer, and it is shown to be both sound and complete.

The multi-agent scenario is not multi-agent in the classical sense where multiple agents are acting in the same environment, but instead considers the planning agent as the only acting agent. In this scenario it is possible to reason about a course of action that depends on both the knowledge about the environment and the knowledge about the knowledge of other agents in it. It is as a result possible to express planning problems where other agent's knowledge must be kept the same, increased (less uncertainty), or lessened (more uncertainty). This is illustrated by the prisoner and guard example, where the guard must be kept uncertain about whether the prisoner has escaped or not.

The planning algorithm is based on the algorithm proposed by Andersen, Bolander, and Jensen [1], which considered strong solutions in a single-agent scenario. Their work uses the internal perspective, and this perspective has in this thesis been extended to the multi-agent scenario. Their work further allows for plan verification by translation of plans into epistemic formulas. Plan verification

has not been treated in this thesis, and the plan representation has instead been changed to policies that naturally captures conditional plans with cycles.

Further work includes using dynamic epistemic logic for re-planning in contrast to offline planning as considered in this thesis. This can make the true multi-agent scenario more feasible, by not having to consider all acts other agents might perform.

The last part of the thesis describes the implementation of the proposed planning algorithm. The implementation was tried with the prisoner and guard example, and successfully found and visualized a solved policy graph. Further work is to find, or defined if such as not been defined before, a proper set of benchmark criteria for planning problems with this type of expressiveness.

Chapter 8

Appendix

```
type agent = string
type world = integer
type model =
{
  W = world list
  A = agent list
  R = (agent,world,world) list
  V = (world, prop list) list
}
type state =
{
  M = model
  Wd = world list
}
```

```
type event = string
type sub = (prop, formula) list
type eventmodel =
{
  Name = string
  Events = event list
  Preconditions = (event,formula) list
  Substitutions = (event, sub) list
  Relations = (agent,event,event) list
}
type action =
{
  E = eventmodel
  Ed = event list
}
```

Figure 8.1: Implementation types of the epistemic state and action. Type `prop` and `formula` can be found in figure 8.2 on the following page in appendix.

```
type prop = string
type formula =
  | bracket of formula
  | conj of formula list
  | disj of formula list
  | prop of prop
  | knows of (agent,formula)
  | neg of formula
  | top of boolean
```

Figure 8.2: Implementation of the formula type.

Algorithm 3 Expansion of node n: `Expand(n)`

for each *action* in *actions* **do**
  let $s \leftarrow$ `productUpdate`$(n.s, a)$
  let m be a new AND-node with $m.s = s$
  let nm be a new edge with $nm.e = a$
  add nm to outgoing edge of n
  add nm to ingoing edges of m
  add m to *AllNodes*
  let $S \leftarrow$ `internalStates`(s)
  let M' be a new list
  let E' be a new list
  for each s' in S **do**
    let mm' be a new edge
    add mm' to outgoing nodes of m
    add mm' to E'
    for each *node* in $allNodes$ **do**
      if *node* is *OR-node* and `existBisimulation`$(node.s, s')$ **then**
        $m' = node$
      end if
    end for
    if m' is undefined **then**
      let m' be a new OR-node with $m'.s = s'$
      add m' to $AllNodes$
      add m' to M'
    end if
    add mm' to ingoing nodes of m'
  end for
  `UpdatePolicies`(n, M', E', m, nm)
  if there exists an α solved policy graph **then**
    return the policy graph
  end if
end for

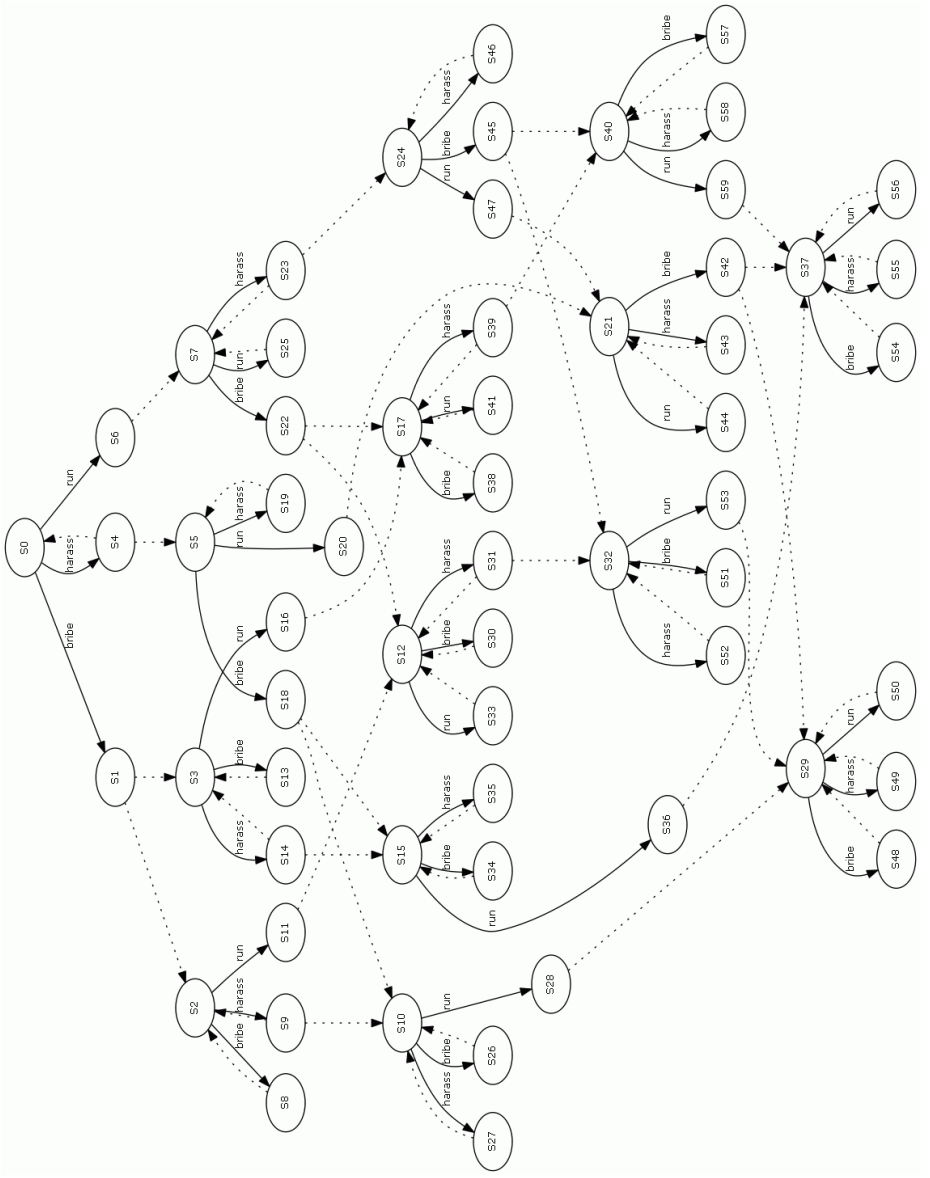


Figure 8.3: The fully saturated planning graph from the prisoner and guard examples, when the goal formula is $\bot$. Nodes are shown without their internal states, and no goal nodes exist.

Bibliography

[1] M. Birkegaard Andersen, T. Bolander, and M. Holm Jensen. Conditional epistemic planning. In *JELIA*, pages 94–106, 2012.

[2] E. Pacuit B. Löwe and A. Witzel. Del planning and some tractable cases. In *Proceedings of the Third international conference on Logic, rationality, and interaction*, LORI'11, pages 179–192, Berlin, Heidelberg, 2011. Springer-Verlag.

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

[4] O. Buffet D. Bryce. 6th international planning competition: Uncertainty part. In *6th International Planning Competition: Uncertainty Part*, August 2008.

[5] M. Ghallab D. Nau and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[6] W. Hoek H. Ditmarsch and B. Kooi. *Dynamic Epistemic Logic*. Springer, P.O. Box 17, 3300 AA Dordrecht, The Netherlands, 2008.

[7] F. B. Bastani J. Fu, V. Ng and Y. I-Ling. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1949–1954, 2011.

[8] R. Jussi. Complexity of planning with partial observability, 2004.

[9] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, December 1987.

[10] AT&T Labs Research. Graphviz.

[11] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[12] F. Sietsma. Model Checking for Dynamic Epistemic Logic with Factual Change. Master's thesis, UvA, CWI, Amsterdam, The Netherlands, 2007.

[13] J. van Eijck. Demo - a demo of epistemic modelling, 2007.

[14] Anders Witzel. modelchecker, 2012.