Conditional Partial Plans for Rational Situated Agents Capable of Deductive Reasoning and Inductive Learning

PhD Thesis

Sławomir Nowaczyk

Department of Computer Science
Lund University, Sweden

Lund, May 2008

Sławomir Nowaczyk
Department of Computer Science
Lund University Box 118
S-221 00 Lund
Sweden

E-mail: Slawomir.Nowaczyk@cs.lth.se
Webpage: http://www.cs.lth.se/home/Slawomir_Nowaczyk/

Printed in Sweden

ISBN 978-91-628-7515-2
ISSN 1650-1268
Dissertation 24, 2008
LU-CS-DISS: 2008-1

Abstract

Rational, autonomous agents that are able to achieve their goals in dynamic, partially observable environments are the ultimate dream of Artificial Intelligence research since its beginning. The goal of this PhD thesis is to propose, develop and evaluate a framework well suited for creating intelligent agents that would be able to learn from experience, thus becoming more efficient at solving their tasks.

We aim to create an agent able to function in adverse environments that it only partially understands. We are convinced that symbolic knowledge representations are the best way to achieve such versatility. In order to balance deliberation and acting, our agent needs to be *time-aware*, i.e. it needs to have the means to estimate its own reasoning and acting time.

One of the crucial challenges is to ensure smooth interactions between the agent's internal reasoning mechanism and the learning system used to improve its behaviour. In order to address it, our agent will create several different conditional partial plans and reason about the potential usefulness of each one. Moreover it will generalise whatever experience it gathers and use it when solving subsequent, similar, problem instances.

In this thesis we present on the conceptual level an architecture for rational agents, as well as implementation-based experimental results confirming that a successful lifelong learning of an autonomous artificial agent can be achieved using it.

Contents

Acknowledgements

There are lots of people I would like to thank for a huge variety of reasons.

First and foremost, I would like to thank my supervisor, Jacek Malec. I could not have imagined a better advisor, mentor and friend for me during these years, and without his amazing mixture of giving me complete freedom and controlling me closely, I would never have finished.

Moreover, I hereby thank all the people in Department of Computer Science at Lund University for the casual yet stimulating atmosphere they maintain here.

I thank all my friends and my family for their support and for always being there for me. Numerous people should be mentioned here, but I will confine myself to naming only five. My parents and my sister Alicja who have been with me my whole life, or at least most of it. Barbara, who made Sweden a much more familiar place than it would otherwise be. And finally Ewa, not only for all the work she did helping me make this document at least a little bit readable.

Last but not least, I express my greatest gratitude to my wife Marlena for her neverending love, unfading trust and unhesitating encouragement throughout all the years we are together.

Chapter 1

Introduction

The principal goal of this PhD thesis is to propose, develop and evaluate a framework well suited for creating intelligent agents — both physical and virtual ones — that would be able to learn from experience, thus becoming more efficient at solving their tasks.

We are especially interested in the development of rational agents that are both autonomous and situated — agents that have their own goals and actively pursue them while being able to survive and prosper in a partially unknown, dynamic and uncontrollable environment.

1.1 Research Idea

An important observation which lies at the very foundation of our approach is that such agents not only need to be modelled as having limited computational resources. They also need to be aware of their own limitations and take them into account (this necessity is presented by Chong and others in [COOP02] in an unusually brilliant and entertaining way). In particular, being placed in a dynamic environment means that assumptions of infinite computational power, inexhaustible memory, etc. are not applicable and even counterproductive. In our project we investigate rational agents that are able to consciously alternate between reasoning and acting and that complement their deductive abilities with knowledge extracted from experience.

It is our belief that a truly intelligent agent needs to use deductive reasoning in order to take advantage of whatever domain knowledge it has been provided with. It also needs to perform inductive learning in order to benefit from experience it has gathered in the past, and to correct missing or inaccurate parts

of its knowledge. Finally, the agent must acknowledge that both reasoning and acting takes time, and consciously put effort into balancing those activities in a rational way.

Clearly, all realistic agents must have limited resources, both mental (CPU power, memory size, reasoning capabilities) and physical (capabilities of sensors and actuators). For quite some time these limitations have been consciously set aside, since ignoring them made it possible to employ a number of important simplifications and to develop useful formal models. We strongly believe, however, that it is now time to revise these limitations and to attempt to create a methodology for creating truly rational, situated and realistic agents in a systematic way.

Despite the significant progress that has been made in several fields of Artificial Intelligence during the last few years, this is still a challenging and ambitious goal. Nevertheless, while not claiming to solve the problem completely, this thesis presents a viable approach that promises to scale towards practical problems.

1.2 Agent Foundations

Rational, autonomous agents that are able to achieve their goals in dynamic, only partially observable environments are the ultimate dream of Artificial Intelligence research since its beginning. Quite a lot has already been done towards achieving that dream, but dynamic domains still remain a major challenge for autonomous systems. In particular, nontrivial environments that are only partially observable pose problems that are beyond the current state of the art — except possibly when dedicated, hand-crafted solutions are developed for very specific tasks.

One of the most promising and successful ways of coping with uncertainty and lack of knowledge about one's current situation is to exploit previous experience. There are numerous ways in which such experience can be gathered and organised, including memorising, creating various mental models, or calculating appropriate probability distributions over possible courses of events, to name just a few.

In our work, one of the biggest challenges is to ensure smooth interactions between the agent's internal reasoning mechanism and the learning system used to improve its behaviour. We investigated a number of formalisms, looking for one which would allow, on the one hand, the reasoner to incorporate knowledge about its own bounded resources (such as reasoning time, memory, sensing

capabilities), and on the other to allow its knowledge and beliefs to be used for inductive learning.

We have decided to base our agent on the idea of conditional partial plans and to have it use a symbolic deductive reasoner employing Active Logic. In particular, we combine those two with ideas from Situation Calculus. Furthermore, we use Inductive Logic Programming algorithms as means of generalising the agent's experience.

A major assumption which drives our research is the idea that realistic rational agents need to live in domains where complexity prevents them from finding complete solutions. Due to limited resources and the necessity to stay responsive in a dynamic world, situated agents are unable to generate a complete plan for achieving their goals. An example of such a domain is the RoboCup setting, where the agent aims to "score a goal", but no complete plan for doing that can be generated (except in some very specific situations, such as being alone, with a ball, and in front of the opponent's goal). Normally, the agent must, at all times, remain open to new observations and adjust its actions accordingly.

In theoretical AI a common solution to a goal satisfaction problem in dynamic domains is to create a *conformant plan*, i.e. a plan which contains provisions for any possible sequence of external events and observations. Such plan can then be proven to reach the goal in any imaginable scenario. For situated agents, however, not only the task of *creating*, but even a requirement to simply *store* such plan could easily exceed the available resources. In realistic domains it is not uncommon at all for an infinite number of different exceptional events to be possible. For example, in a simple process of driving home from work, a person could encounter any number of traffic accidents, but it is counterproductive to prepare in advance for hundreds of them.

1.3 Conditional Partial Plans

With such considerations in mind, we have decided that it is not realistic to expect our agent to find a complete plan that achieves its goals. Throughout this thesis, we will mainly use nondeterministic domains (or ones which appear to be nondeterministic to the agent due to its incomplete understanding of them) to illustrate this phenomenon.

For this reason we have decided that our agent needs to create and reason about *partial plans*. By that we mean plans that (hopefully) bring it somewhat closer to achieving the goal, but which are manageably simple and short enough to be computable in reasonable time. It is crucial to point out that, in general,

there exists no way in which an agent could actually be *certain* that executing a particular partial plan is, indeed, bringing it closer to achieving its goal (other than expanding this plan until it becomes a complete one). The agent is even less likely to be able to decide with certainty whether a partial plan is an optimal one to execute or not.

Therefore, our focus in this work is not for an agent to strive towards generating optimal solutions, but to make rationally motivated decisions to settle for behaviours that are "good enough" instead. Clearly, once a dynamic environment and time constraints are taken into consideration, the optimality of a solution can no longer be realistically expected.

At the same time, in order to make informed decisions about balancing deliberation and acting, the agent needs to be *time-aware*, i.e. it needs to have the means to estimate its own reasoning and acting time. Moreover, it needs to be able to influence them — at the very least it needs the ability to decide, in the middle of solving a problem, that it has had enough and to "give up". Only after the agent is able to estimate the progress of its own deliberation, does it have the means to consciously balance reasoning and acting.

We intend, therefore, for the agent to judge by itself whether it is more beneficial to begin executing one of the existing plans immediately or rather to continue deliberation and, possibly, develop longer and more complete plans in order to avoid making an unrecoverable mistake. In other words, the agent will be performing on-line planning and interleaving it with plan execution.

In this context it is especially important to distinguish a special class of actions and plans, so called "information-providing" ones, which allow an agent to acquire additional knowledge about the world. Such plans are continuously in the centre of our attention, since we believe that they are instrumental in addressing the problem of the limited resources our agent has.

These information-providing plans allow the agent to "cheat" away part of the complexity of the external world, especially when planning activity is interleaved with their execution. In particular, performing them at the right time allows the agent to greatly simplify its subsequent reasoning process — it no longer needs to take into account the vast number of possible situations that are inconsistent with the newly observed state of the world. Thus, it can proceed further in a more effective way, by devoting its computational resources to more relevant issues. Coincidentally, this is also how humans approach unsolvable problems.

Therefore, situated agents must consciously alternate between reasoning, acting and observing their environment, and in some cases do all of this at

once. We aim to achieve this by making the agent create short partial plans and execute them, learning more about its environment throughout the process. The agent also generalises its own experience to evaluate the likelihood of any particular plan leading to the goal.

In addition to being partial, the plans our agent reasons about are *conditional*, which means that actions to be taken depend on observations made during execution. This is especially important when the agent attempts to learn from experience and to evaluate plans, deciding that some of them are better than others. Interestingly, in many domains sequential plans are not sufficiently generic to allow such evaluation to be successful, since their usefulness is heavily grounded in the current situation.

Conditional plans, on the other hand, are often more universal and their quality can be estimated more meaningfully, as we will explain in Section 2.3 and in Chapter 6. It is often the case that, if agent's knowledge about the world is limited (i.e. it cannot predict the effects of its own actions with certainty and most actions can lead to various unpredictable observations), only very short sequential plans can be classified as good ones, since anything longer risks neglecting new and potentially important information. At the same time, a conditional plan may be much more complex since it can contain provisions for unexpected events.

1.4 Reasoning Mechanism

It is our belief that deductive knowledge, at least in many of the domains we are interested in (RoboCup, driving home from work, Wumpus game), may contain more details and be more accurate than other forms of representation (such as numerical or probabilistic ones). Our agent needs to be able to handle non-stationary, adverse environments, to cooperate with others in multi-agent settings and to plan for goals more complex than simple reachability properties (such as temporally extended goals and restoration goals). Our intent is to create an agent able to function in a hostile environment that it can only partially observe and that it only partially understands.

We are firmly convinced that symbolic knowledge representations are the only way to achieve such versatility. However, the requirements we presented in the previous two sections put rather unique constraints on the reasoning mechanisms employed by such an agent.

We have decided to use Active Logic [EKM$^+$99] as the main logical formalism for our agent, since it well suited to solve several of the problems we

face. At the same time, it is sufficiently flexible to allow us to incorporate extensions that will deal with the rest. Active Logic was designed for non-omniscient reasoners and we consider it a good reasoning technique for versatile agents. Its main distinctive feature is the characterisation of deduction as an ongoing process instead of focusing on a fixed point of entailment relation.

A critical feature of Active Logic, from our point of view, is its ability to model different kinds of introspection very well. The ability to reason about its own knowledge or lack thereof is an absolutely crucial capacity for our agent. It must be able to decide that it needs to observe certain features in the environment and to choose appropriate actions to do that. To this end, we combine Active Logic with some ideas from Situation Calculus, which allows an agent to reason about the current state of the world, as well as about how it will change as a result of executing a particular plan.

At this time, the research field of planning has matured enough that exploring new, more ambitious settings is feasible. This should allow us to bring artificial agents closer to what humans are capable of. Several researchers are investigating settings beyond the so called "classical planning" paradigm, and our own work fits well within this trend. It is crucial to note, however, that planning itself — while definitely important — is only one aspect of our solution. We do not focus so much on the generation of plans itself, but rather on ways to combine it with other areas of Artificial Intelligence.

As explained before, our goal is to create an agent which is able to function in adverse environments that it can only partially observe and that it only partially understands. Therefore, we need to provide it with some important advantage, or else its task will be impossible to perform. To this end, we designed our agent so that it lives through a large number of episodes, each similar but not identical to the previous one, in order to learn from its mistakes and to improve its efficiency. This way the agent can start with limited knowledge and understanding of the domain, but with time it will become more proficient — each time it tries to solve a problem, it will do so in a more efficient way. The idea is to have a system that is usable (even if inefficient) early on, but which achieves the desired quality given more time.

The experience gathering can be done in many ways, and it is interesting if the agent itself is aware of those differences. We can imagine a system when the training phase is performed in a simulator, and the agent needs not to be "afraid" of trying new things, since even if the outcome is bad, there are no long-lasting effects of its mistakes. Similarly, the training can also be done in a physical world, which will likely be more accurate, but which carries with

it the danger of permanently destroying the agent — especially when it is not done in the safety of a lab, but rather in real environment (for example, a Mars rover). Therefore, the agent needs to be much more "cautious" when attempting potentially dangerous actions.

1.5 Illustratory Domain

Throughout this thesis we will be using a simple game called Wumpus, the well-known testbed for intelligent agents introduced in [RN03] to better illustrate our ideas. The game is very simple, easy to understand, and people have no problems playing it effectively as soon as they learn the rules.

For artificial agents, however, this game — and other similar applications, including many of practical importance — still remains a serious challenge. The main reason for that is the combinatorial explosion of game states that need to be considered. It is impossible to analyse all of them explicitly, and while symbolic reasoning shows promise to provide effective generalisations over them, at the current state of the art one still needs to test ideas and prototypes on small, artificial examples.

The Wumpus game is played on a square board. There are two characters, the player and the Wumpus. The player can, with each turn, move to any neighbouring square, while the monster does not move at all. The position of the beast is not known to the player, he only knows that it hides somewhere on the board. Luckily, Wumpus is a smelly creature, so whenever the player enters a square, he can immediately notice if the creature is in the vicinity. The goal of the game is to find out the exact location of the monster — this is done by moving across the board and checking which squares smell. At the same time, if the player enters the square occupied by the monster, he gets eaten and loses the game.

This is only the simplest version of the Wumpus game — a number of different variants have been introduced throughout the years, both as research illustrations and as actual games intended to be played by people. The most common variation include moving Wumpus, additional traps such as pits, additional monsters such as Bats, using a maze instead of a simple board, equipping the player with a bow which can be used to kill the Wumpus, hiding treasure for the player to find, etc.

Our research, obviously, is not designed with the Wumpus game specifically in mind. We use it only as an illustratory domain to explain our ideas on. This game has a number of different aspects that make it especially appropriate for

presenting our approach. It is very simple both to present and to formalise, while it has a high number of observations to be made (which makes conformant plans uninteresting solution). Additionally, the perfect information version of the game (when the agent *knows* Wumpus's position from the very beginning) is extremely simple, which means it does not distract our attention.

For some of our learning experiments we also use a second domain, which we later call Chess for short, a modified version of the "king and rook *vs* king and knight" chess ending. This game is based on the normal rules of chess, but there are only four pieces on the board: both kings, white rook and black knight. It is interesting to note that depending on the initial position of all those pieces (assuming an optimal play by both players), the game will either result in a draw, or white will win.

In our case, since we are interested in partially unknown environments, we have modified the rules in such a way that the agent does not know how the opponent's king is allowed to move — *a priori*, any move is legal. This means that an agent can never find a winning strategy using pure deduction. Our agent will need to use learning to discover what kinds of moves are actually possible and that it can, in fact, succeed.

These two domains are, of course, only examples and the architecture presented here does not depend on them. In order to better understand the goal of our research, it can be helpful to imagine a setting similar to the *General Game Playing Competition* [GLP05]: our agent is given some declarative knowledge about a domain and is supposed to act rationally from the very beginning, hopefully becoming more and more proficient as it gathers more experience.

1.6 Interdisciplinary Aspect

Despite multiple attempts, both past and ongoing, the vast majority of AI research is being done in specialised subfields and it is our belief that none of these subfields *alone* can give us truly intelligent, rational agents. Our architecture, which to the best of our knowledge is novel, may be one way to integrate a number of such approaches.

One of the major contributions of this thesis is its interdisciplinary aspect, namely the fact that it brings together solutions from several different subfields of Artificial Intelligence and shows how they can be combined in a coherent, well-performing system. Most of the specific solutions used in this work (for example planning and learning algorithms) are not quite state of the art implementations in their respective areas, but have been chosen for simplicity of

presentation or popularity rather than for pure performance. Nevertheless, the architecture of our agent is modular and more efficient solutions can easily be substituted at any time.

Throughout this thesis we will be using a number of examples based on the previously introduced game of Wumpus. The main problem in this game is to learn the position of the monster. In order to plan for achieving this objective, an agent needs to be able to reason about its own knowledge and about how will it change as a result of performing various actions. Thus, the logic it utilises in its reasoning needs to strongly support epistemic concepts. At the same time, a notion of time-awareness is necessary, as we require our agent to consciously balance planning and acting.

To accommodate these requirements, we employ a variant of Active Logic [EKM+99] as the agent's underlying reasoning apparatus. This logic was designed for non-omniscient agents and has mechanisms for dealing with uncertain and contradictory knowledge. We believe it is a good reasoning technique for versatile agents, as it has been successfully applied to several different problems, including some in which planning plays a prominent role [PPT+99].

The domain of the Wumpus game, in several of its more complex variants, has one more interesting feature, namely that the desired behaviour of the agent consists of two phases. First, it has to gather some information ("Where is the Wumpus?") and, after that, it needs to exploit this knowledge ("How to kill it?"). At the same time, this distinction is not present in the rules at all — it is something the agent will need to discover on its own. This problem is rather difficult one, and there is an active field of research dealing with it, under various names, the most commonly used being "subgoal discovery".

To summarise, our agent will create several different plans and reason about the potential usefulness of each one — including what knowledge can be acquired by executing it. Further, it will judge whether it is more beneficial to immediately begin executing one of those plans or rather to continue deliberation. In our approach, the agent continuously reasons about the world, enriching its knowledge using both observations and deduction.

Moreover, we expect the agent to live significantly longer than the duration of any single game episode, so it should generalise whatever solutions it finds. In particular, the agent needs to extract domain-dependent control knowledge and to use it when solving subsequent, similar, problem instances.

Due to resource limitations, our agent needs to select only the *most relevant* subset of its own knowledge to be used in learning and for generalising its experience. In particular, it is not practical to expect ILP algorithms to be able

to find useful generalisations among vast amounts of unrelated and redundant knowledge.

All of the features mentioned above have been extensively studied in Artificial Intelligence literature, including ideas on how to integrate various combinations of them — we will discuss some of this work in the next section. However, to the best of our knowledge, nobody has yet attempted to merge all, or even most, of these features together into one, consistent framework.

1.7 Overview of the Research Area

In this section we present a short, necessarily selective overview of several research areas relevant to this work. In particular, we are going to discuss agent architectures, reasoning with limited resources, planning and learning as well as, particularly important in this context, their overlaps.

The topic of this thesis is how to create autonomous, rational, situated agents capable to learn. In a sense this is the goal of the whole field named *Artificial Intelligence*, although it has been expressed using different words in different time periods. The term *agent* has become very popular in the middle of the previous decade, with an excellent textbook by Russell and Norvig [RN03] being considered now as "the agent-based introduction to AI". A comprehensive collection of the foundational articles covering the topic can be found, for example, in [WR99].

The work presented here is related to studies of architectures for general intelligence. There have been many attempts to define such an architecture, with SOAR being the most prominent, most successful and probably the oldest of them all ([Lew01] provides a general introduction). In contrast to SOAR, we do not claim any cognitive plausibility of our architecture, focusing our interests mainly on the task of achieving rational behaviour of an artificial agent.

There is a number of architectures that have been successfully used to develop agents, including real physical systems which demonstrate large degrees of autonomy, situatedness and, in some cases and to some extent only, also rationality. Chronologically, among the first ones were e.g. NASREM [AML89], Reactive Action Packages [Fir89], the subsumption architecture [Bro91], InterRAP [MPT95], Procedural Reasoning System [GL87], and ATLANTIS [Gat91]. Later on, a number of good overviews have appeared, e.g. chapter six of Arkin's textbook [Ark98] (pointing to the fact that "the nature of the boundary between deliberation and reactive execution is not well understood at this time, leading to somewhat arbitrary architectural decisions", p.

207), Jennings, Sycara and Wooldridge [JSW98], Müller [Mül99], Lee [Lee00], and Kortenkamp et al. [KBM98] to mention just a few.

Our approach is quite different from the layered architectures mentioned above. In particular, we do not focus specifically on the reactive part of the system, hiding it as just one part of the Actor module. This does not mean that we diminish the necessity or importance of reactivity, but rather that we simply decided to concentrate on the higher-level reasoning aspect of our agent as it is less understood and requires more attention. We make sure, however, that those higher levels of our architecture remain sufficiently flexible to be able to handle the requirements of the reactive part.

Although we stress the need for rationality and the importance of reasoning, our approach also differs from the one exhibited by Beliefs-Desires-Intentions systems and BDI architectures (see, among others, [Woo00] for a formalised approach to this topic). BDI systems are particularly good for specifying situated intelligent systems, less so for implementing them. In our work we do not explicitly distinguish intentions, treating them in a manner very similar to goals. This is, however, due to the fact that BDI approaches usually lack a learning component, which is the central issue in our approach, and that allows an agent to account for its intentions in a different, yet equally effective way.

The relevant issues of formal analysis of agent architectures has not attracted much attention until very recently. An example of correctness analysis of the classical PRS programs may be found in [Wob00], in its turn based on an early work by Rao and Georgeff [RG93], while a more general look at the control structures of rule-based systems, relevant from the point of view of our approach, may be found in [HBHM99]. However, most literature on this topic appears to be rooted in the database theory, where active real-time databases have been studied formally for a while.

Finally, the issue of comparing agent architectures, especially with respect to their effectiveness and suitability for intended applications, is an important topic worth paying attention to and undoubtedly requiring further studies. An interesting preliminary discussion may be found in [HHM$^+$00] and [Lee00], although conclusive results are yet to be obtained.

Please observe that the main notion our agent reasons about is its own knowledge about the world, as opposed to plainly thinking about the world around it. There is a large research area of *metareasoning* focusing on providing formal tools for letting agents do just that. However, those formal systems almost universally focus on idealised reasoners and are thus incapable of capturing the limitations of real agents.

One of the main reasons for problems with describing resource-bounded reasoning is that the formal systems used for this purpose are too powerful. Quite often such a system is based on some propositional or first-order language extended with a modality denoting belief. This immediately leads to the *omniscience problem*: use of any *normal* modal logic equipped with the **K** axiom $\models \Box(\alpha \rightarrow \beta) \rightarrow (\Box\alpha \rightarrow \Box\beta)$ and the necessitation rule $\frac{\alpha}{\Box\alpha}$ will force the agent using it to be explicitly aware of all logical consequences of its current beliefs (see, e.g., [FHVM95]). This means that its set of beliefs will necessarily be infinite and it must always be consistent: apparently a very non-realistic assumption in the case of bounded reasoners.

There exist a number of approaches that try to deal with the problem of omniscience. Speaking generally, any such solution must address the need to model bounded resources of agents and, independently, its incomplete reasoning mechanisms. The solutions might be roughly classified into four groups: weakening the system by using a non-standard semantics; formally distinguishing explicit and implicit knowledge; removing closure properties; and syntactification. The first two still suffer from partial logical omniscience [Ågo04], although they constitute a step in the right direction.

A good example of the second line of thought is the *logic of implicit and explicit belief* proposed by Levesque [Lev84]. His idea consists of distinguishing *explicit* beliefs, i.e., those currently available explicitly in agent's knowledge base; and *implicit* beliefs, i.e., those entailed by explicit beliefs, but not (yet) derived. In order to be able to handle that distinction, the semantics of the classical epistemic logic must be modified. According to Levesque, the usual possible worlds semantics is too coarse-grained, while simple sets of formulae are too fine-grained of a choice. His semantics is based on *situations* that are subsets of possible worlds. Unfortunately, partial omniscience makes this system inappropriate for describing truly limited agents.

The approach of Fagin, Halpern, Moses and Vardi [FHVM95], *Interpreted multi-agent systems* captures the evolution of an agent's knowledge in a nice way. The system consists of the usual knowledge/belief modalities (K_x, indexed by multiple agents involved), mixed with the classical temporal operators (Eventually $\Diamond$, Always $\Box$, Next $\bigcirc$, Until $\mathcal{U}$). The resulting system is interpreted on so called *runs*. The system is still too powerful for our needs, although it may be modified in the direction of non-omniscient agents. The same remarks apply to the recent proposal of van der Hoek and Wooldridge, *Alternating-time Temporal Epistemic Logic (ATEL)*, [HW03], in which the usual knowledge/belief modalities (K_x) and the classical temporal operators are extended

with a dynamic-logic-like concept of *cooperative actions*. The interpretations are based on *concurrent game structures*. Although the authors mention the possibility of describing non-omniscient agents, the main system is developed for at least partially omniscient entities. A similar system, based on active logic (see below), has been proposed by Grant, Kraus and Perlis [GKP00].

The last system we would like to mention in this context is the *Logic of Finite Syntactic Epistemic States* proposed recently by Ågotnes [Ågo04]. His system is based on ATEL, but does not assume any structure in the underlying language — the epistemic states of an agent are purely syntactical structures. Knowledge evolution mechanisms are modelled using *rules*; they do not necessarily need to be sound or complete. Although appealing from the formal point of view, this system does not provide any hints about dealing with the computational complexity of the problem.

An approach resembling anytime algorithms [ZR93] but applied to the area of deduction has been proposed by Fisher and Ghidini [FG00]. They provide a logical system capable of adapting its deductive power to the resource constraints. However, no useful bounds (at least for us) can be derived from this approach — the only guarantee is that the number of theorems will be smaller in some cases, but the proofs can still be very long and no fixed limit in terms of the number of steps may be given.

The attempts to, in a principled way, constrain the inference process performed in a logical system have been done as long as one has used logic for knowledge representation and reasoning. One possibility is to limit the expressive power of first-order logical calculus (as, e.g., in description logics) in order to guarantee polynomial-time computability. There is a number of theoretical results in this area (see e.g [Ebb99]) but we are more interested in investigations aimed at practical computational complexity. One of the more popular approaches is to use a restricted language (again, like description logics), see [GINR99, Pat85, Pat86] for examples of this approach in practice.

Another possibility is to use polynomial approximations of the reasoning process. This approach is tightly coupled with the issue of theory compilation. The most important contributions in this area are [SK96, CD97, CS92, GPS98]. However, this approach, although it substantially reduces the computational complexity of the problem, still does not provide tight bounds on the reasoning process.

Yet another possibility is to constrain the inference process in order to retain control over it. An early attempt has been reported in [Lev84]. The next step in this direction was the step-logic [Elg88] that evolved into a family of

active logics [EKM+99]. Such a restriction is actually a reasonable first step towards developing a formal system with provable computational properties. Active logics have been used so far to describe a variety of domains, like planning [PPT+99], epistemic reasoning [Elg91], reasoning in the context of resource limitations [NKP93] or modelling discourse and dialog. In this work we are using active logic for deduction involving the temporal aspect of metareasoning.

A similar idea was introduced in [PB04], where authors investigated how various actions and observations of their effects modify an agent's belief state. They describe how such modifications can be propagated backwards and forwards through the state history: as the agent gains new knowledge, it can infer that various statements *did* hold in past states of the world, even if it did not know it *then*. Authors also show how such propagation can be used for dealing with temporally extended and restoration goals.

There is a growing insight that logic, if it is to be considered a useful tool for building autonomous intelligent agents, has to be used in a substantially different way than before. Active logics are one example of this insight, while other important contributions might be found, e.g., in [GW01] or [WL01].

The second major area of AI touched upon in this work is planning. There exist comprehensive recent monographs on this topic (see e.g. [GNT04]). The field evolves very dynamically, with state of the art planning systems solving problems of large complexity; c.f. the yearly conference on automatic planning and scheduling ([LSBM06, BFT07] are the two most recent ones) and the associated contests of planning systems. A lot of effort is devoted to incorporating (possibly by compiling) domain knowledge into planning systems in order to improve search efficiency.

Another track of research focuses on (deductive) planning, taking into account the incompleteness of the agent's knowledge and its uncertainty about the world. Conditional plans, generalised policies, conformant plans and universal plans are the terms used by various researchers [CRB04, PB04, HW02, BCT04] to denote the, in principle, identical idea: generating a plan which is "prepared" for all possible reactions of the environment. This approach has much in common with control theory, as observed in [BG01] or earlier on in [DW91].

Research that would attempt to integrate learning into this approach is still in its infancy, with just a few published results to date. One of the first completed systems was XFRMLEARN by Beetz [Bee02], building on XFRM system of McDermott [McD92]. However, even there the stress is put on the reactive behaviour of the agent (in this case the autonomous robot Rhino) more than

on the symbolic plans necessary to achieve intelligence. An even more reactive attempt is reported in [LK02], where the planning addressed is just shortest path search, although the concept of *lifelong planning* is very similar to the idea explored in this thesis.

For a well-developed discussion of conditional partial plans and interleaving planning and execution see for example [BCT04], where the authors introduce the notion of a *progressive plan* — intuitively, one that provably moves the agent closer to the goal. They also present an algorithm for finding such plans in a nondeterministic but fully known domain and prove that it is guaranteed to find a solution if one exists. Another example of research on interleaving planning with execution is based on early work on real-time search incorporating learning (LRTA*, [Koe01]) although, as many other works in this spirit, it focuses on path-planning: a very limited kind of reasoning.

In a sense, our treatment of plans in this work is related to the notion of hierarchical planning, since the conditional partial plans we consider are very similar to macro-operators [GNT04]. Our goal is to let the agent learn which conditional partial plans are *good* to later use them as building blocks for finding complete solutions.

A somewhat similar, very interesting idea was pursued in [Nyb05], where the author uses a classical planner to plan for the "optimistic" case, where an agent can choose the most favourable outcome of each non-deterministic action. From such an optimistic plan it is then possible, using knowledge of probabilities of each action outcome, to generate more realistic plans by updating relative costs of optimistic actions.

There have been significant amounts of work done in the machine learning area about what actions to take in a particular situation. The methods may be divided into model-free ones and those based on models, either available or discovered on-line. The first group leads to reinforcement learning, where an agent learns policy, i.e. appropriate action for every possible state of the world. Sutton [Sut90] did one of the early examples of work in this direction, explicitly naming architecture for learning, planning and reacting. However, compared to ours, this architecture neglects the declarative knowledge and symbolic reasoning completely. For an overview of reinforcement learning cf. Barto and Suttman's textbook [SB98].

In the case of methods based on explicit, symbolic models the first to mention is probably [DF95], who presented results establishing conceptual similarities between explanation-based learning and reinforcement learning. In particular, they discussed how Explanation-Based Learning can be used to learn action

strategies and provided important theoretical results concerning its applicability to this aim.

One notable example of this track of research is [Kha99], where the author showed important theoretical results about PAC-learnability of action strategies in various models. In [Moy02] the author discussed a more practical approach to learning Event Calculus programs using Theory Completion. He used extraction-case abduction and the ALECTO system to simultaneously learn two mutually related predicates ($Initiates$ and $Terminates$) from positive-only observations. Recently, [KL06] developed a system which learns low-level actions and plans from goal hierarchies and action examples provided by experts, within the SOAR architecture. Yet another fresh piece of work close to this approach is documented in [LC06], where *teleoreactive logic programs*, possibly even recursive ones, are used for representing the action part of an agent. On top of that a learning mechanism, quite similar to ILP, is employed to improve the existing action programs.

In the general field of Inductive Logic Programming, there is a large number of systems being developed, such as CLAUDIEN [DRL96], MOBAL [SEKW], Charade [RMM$^+$94], Rulex [AG02] and others. We have based our work on PROGOL mainly because it is the most popular one and various researchers have been working on improving multiple aspects of it, among others [Yam96] and [BS99]. To the best of our knowledge, however, nobody has yet used it to classify conditional partial plans.

Combining planning and learning is an area of active research, in addition to the extensive amount of work being done separately in these respective areas. However, most of the related work we are aware of is devoted to either using state-of-the-art learning in a rather limited planning framework, or to using limited learning in a more complex planning setup. Comparisons of the two areas are also relatively common, while the true, nontrivial combination will apparently require much more investigation. Since we believe it to be very promising, this thesis is aiming to attract attention to this line of research.

One attempt to escape the trap of a large search space has been presented in [DRD01], where relational abstractions are used to substantially reduce cardinality of search space. Still, this new space is subjected to reinforcement learning, not to a symbolic planning system. A conceptually similar idea, but where relational representation is being learned via behaviour cloning techniques, is presented in [Mor04].

Outside the domain of planning, there is a lot of important research being done in the learning paradigm. Recently [CM03] presented several ideas

on how to learn interesting facts about the world, as opposed to learning a description of a predefined concept. A somewhat similar result, more specifically related to planning, has been presented in [FYG04], where the system learns domain-dependent control knowledge beneficial in planning tasks. From another point of view, [KR95, KR97] put forward a framework for learning done "specifically for the purpose of reasoning with the learned knowledge," an interesting early attempt to move away from the *learning to classify* paradigm, which dominates the field of machine learning.

In a sense this is similar to the ideas discussed in [FYG04], where authors use the Markov Decision Process to represent planning domains and approximate policy iteration as means of learning agent's behaviour. They use long random walks to create progressively harder goals, thus bootstrapping the agent in its learning of domain-dependent control knowledge.

An interesting line of research, which possibly could be useful in our case, was presented in [GT04], where authors attempt to deductively generate a domain-specific hypothetical language that is as simple as possible, and yet expressive enough to represent all the necessary concepts in a particular domain. This language is then used by an inductive learning algorithm to create generalised policies from solutions of small problem instances.

Many of the ideas investigated in this thesis have been analysed previously, in numerous disguises. There have been many, to a large extent successful, attempts to attack the specific scientific and practical problems related to autonomous, situated, rational agency. However, attempts to merge them into a single, consistent framework have been very rare and so far incomplete.

1.8 List of Publications

This section contains the list of the author's publications[1] presenting partial results described in this thesis. All of them are available from my home page:
`http://www.cs.lth.se/home/Slawomir_Nowaczyk/`
by following the "publications" link. Similarly, the software, input data sets and settings used in experiments reported in this thesis (mainly in Chapter 8) are available from that page by following the "experiments" link.

In case of the papers co-authored by my supervisor, I affirm that I am the principal author of the work described there.

[1] This section, as opposed to all other text in this thesis, is kept in the first person to stress the statement I make about the authorship of the papers listed below.

- AAAI'06: Sławomir Nowaczyk, Learning of Agents with Limited Resources, AAAI-06 Student Abstract and Poster Program, 2006 — an introduction to the concepts, somewhat obsolete but probably interesting both from a historical perspective and as a general overview [Now06a].
- LRBA'06: Sławomir Nowaczyk, Partial Planning for Situated Agents based on Active Logic, Workshop on Logics for Resource Bounded Agents, ESSLLI 2006 — it predates the implementation of the learning part somewhat, but contains a relatively good description of agent's reasoning mechanisms [Now06b].
- ICMLA'07: Sławomir Nowaczyk, Jacek Malec, Learning to Evaluate Conditional Partial Plans, The Sixth International Conference on Machine Learning and Applications (ICMLA'07), Cincinnati, Ohio, 2007 — it describes the results of learning experiments, in two example domains: Wumpus and Chess [NM07c].
- ABC'07: Sławomir Nowaczyk, Jacek Malec, An Architecture for Resource Bounded Agents, Workshop on Agent Based Computing (ABC'07), Wisła, Poland, 2007 — quite complete description of agent's architecture [NM07a].
- LRBA'07: Sławomir Nowaczyk, Jacek Malec, Relative Relevance of Subsets of Agent's Knowledge, Workshop on Logics for Resource-Bounded Agents in Durham, United Kingdom, 2007 — discussion of the idea of "knowledge relevance" concept and how it affects learning results [NM07d].
- MICAI'07: Sławomir Nowaczyk, Jacek Malec, Inductive Logic Programming Algorithm for Estimating Quality of Partial Plans, 6th Mexican International Conference on Artificial Intelligence, 2007 — adaptation of PROGOL algorithm which better fits the learning task within my architecture [NM07b].

Chapter 2

The Agent

2.1 Introduction

The architecture of our agent (see Fig. 2.1) consists of four main functional modules. Each is responsible for a different part of the agent's rationality, but the overall intelligence is only achievable by the interactions between all of them. At the same time, each of these modules corresponds to an area of active research within the field of Artificial Intelligence. This makes our architecture an interesting exercise in combining different ways of looking at AI.

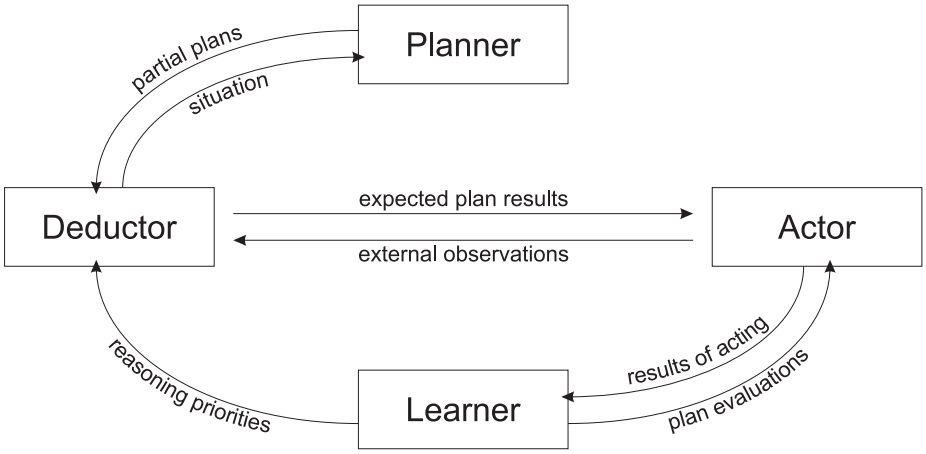

Figure 2.1: Architecture of an agent.

2.2 Agent Architecture

The first of the modules mentioned above is *Deductor*, which corresponds to a typical "core" of a symbolic, logic-based reasoning agent. One interesting twist is that in our framework the logic in question is not the classical one, but a formalism called *Active Logic* instead. This provides Deductor with various means to better interact with other modules, as will be explained later in this text.

The purpose of this module is to perform symbolic reasoning about the world, actions the agent can perform, and the consequences of them. It is designed to analyse facts the agent observes and to deduce as much as possible about the hidden state of the world, especially about the agent's current situation. Furthermore, it predicts — as far as the agent's past experience, an imperfect domain knowledge and limited resources allow — what effects each of the plans under consideration might have, including what new knowledge may be acquired.

In this sense, the Deductor module is the one responsible for classical "thinking". It uses a logical formalism based on a combination of Active Logic and Situation Calculus in order to reason about consequences of the agent's beliefs and consequences of the actions it is considering. Based on domain knowledge and previous observations, it first deduces as much as possible about the current state of the world. Afterwards, it analyses a number of possible plans, predicting what will be the effect of executing each one of them. In particular, it accounts for the fact that some actions may be information-providing ones — it does this by anticipating how agent's knowledge will change. Deductor will be explained in more detail in Chapter 3.

The second module is *Planner*, which generates partial, conditional plans applicable in the agent's current situation. After Deductor finishes analysing results of observations gathered in previous steps, Planner produces a number of plans which are potentially interesting candidates for execution.

The architecture is designed so that it is relatively easy to use any of the state of the art, efficient planning algorithms (such as FF, KACMBP, POND or any other — for an overview of possibilities, see [GNT04]). Still, doing so is not necessarily trivial, since the domain knowledge used by our agent — and especially the results of its reasoning process — is, typically, expressed in a language significantly richer than what classical (or even many non-classical) planners accept. Therefore, the intended mode of operation for our agent is to simplify the domain using a process known as *relaxation*, in order to enable the use of efficient planning algorithms — at the expense of accuracy of obtained

plans. By relaxing domain description in a number of different ways, the agent will generate several different plans, and each of them will be analysed by Deductor. We describe this mechanism, together with other details of Planner, in Chapter 4.

The third main module, *Actor*, supervises Deductor's reasoning process and evaluates plans that Planner has come up with. Taking into account the possible consequences of each course of action, as reasoned about by Deductor, it tries to find out which plan is most worth executing. Also, this module allows an agent to take into account the limitations of its own resources, and it fulfils the requirement of actively and consciously balancing reasoning and acting. Finally, Actor is an overseer of interactions between an agent and its environment.

From another point of view, Actor can be seen as the *reactive* part of the agent. The architecture is designed in such a way that it allows Actor to monitor Deductor and to estimate its reasoning progress. Thus, it can break the deliberation when a particularly interesting plan has been discovered or when it decides that nothing worthwhile is likely to be deduced anymore. At the same time, it observes the external world, analysing and interpreting the agent's sensor data in order to react whenever something interesting happens in the environment. It is responsible for observing the world and for introducing effects of the agent's actions — and, potentially, also other changes of the world — into the agent's knowledge base.

On the one side, Actor watches over agent's reasoning process and makes decisions about when its results are sufficiently well developed to begin being acted upon. On the other, it observes the external world in order to detect events which require the agent's immediate response — a response that can either be a purely reflexive action such as collision avoidance, a simple update of Deductor's knowledge base with new information, or a complete overhaul of the reasoning process.

These three modules form the core of the agent. By creating, reasoning about and executing a sequence of conditional partial plans our agent moves progressively closer to its goal. Finally, it reaches a point where a complete plan can be directly created by Planner, its correctness can be proven by Deductor, and its execution by Actor fulfils the ultimate objectives of our agent.

However, the success of such a scenario depends entirely on whether each partial plan in this sequence is indeed moving an agent *closer* to achieving its ultimate goal. Since the agent might not have enough resources to fully utilise whatever knowledge it possesses and, moreover, this knowledge may be incomplete, there is no guarantee that executed plans will do that. In particular, it is

possible that Actor makes a mistake, or even a series of them, leading to the agent losing the game or reaching a position in which winning is impossible.

It is important to understand that, given two arbitrary partial plans, there is no general way of determining whether one of them is better than the other. There is a number of special cases, but the only fool-proof method of deciding that a partial plan is indeed beneficial is to extend it all the way to the ultimate goal, thus turning it into a *conformant plan*. As we explained in the Introduction, this is not a viable approach in the setting we are investigating.

This is the reason for including the fourth module in our architecture, called *Learner*. After the game episode is over, regardless of whether the agent has won or lost, it inductively generalises the experience gathered — in an attempt to improve Deductor's and Actor's performance. The architecture is designed to use the learned information to fill gaps in the domain knowledge, to figure out generally interesting reasoning directions, to discover relevant subgoals and, finally, to more efficiently choose the best partial plan to be executed.

To this end, the *Learner* module analyses the agent's knowledge, the chosen course of action, its results and subsequent observations. From this experience it induces rules for improving the performance of each of the agent's modules. In this work we primarily focus on using learning to better estimate quality of conditional partial plans. In this sense, results of learning are used both by Deductor and by Actor. In particular, as explained above, it is very difficult to predict whether a particular plan is a step in the right direction or not. Using machine learning techniques is one possible way of achieving this, as investigated in more detail in Chapters 6 and 8.

In principle, learning could take place at any time, but we do not currently see much benefit of learning in the middle of the game. Our variant of the Wumpus game is so simple that a single episode does not last very long, and there is plenty of useful information that is only available to the agent once the game is finished — once all the hidden information is revealed. This information is the most valuable during learning. In other settings, however, a mid-episode learning could very well be justified, yet it would also pose the question of how the agent is to decide when it should be done. In the same manner as it now balances reasoning and acting, the agent would need to balance reasoning, acting and learning.

Generally, the ultimate goal of this architecture is to allow putting together state-of-the-art solutions from several different areas of Artificial Intelligence. Despite multiple efforts, those done in the past and those still in progress, the vast majority of AI research is being done in specialised subfields. While such

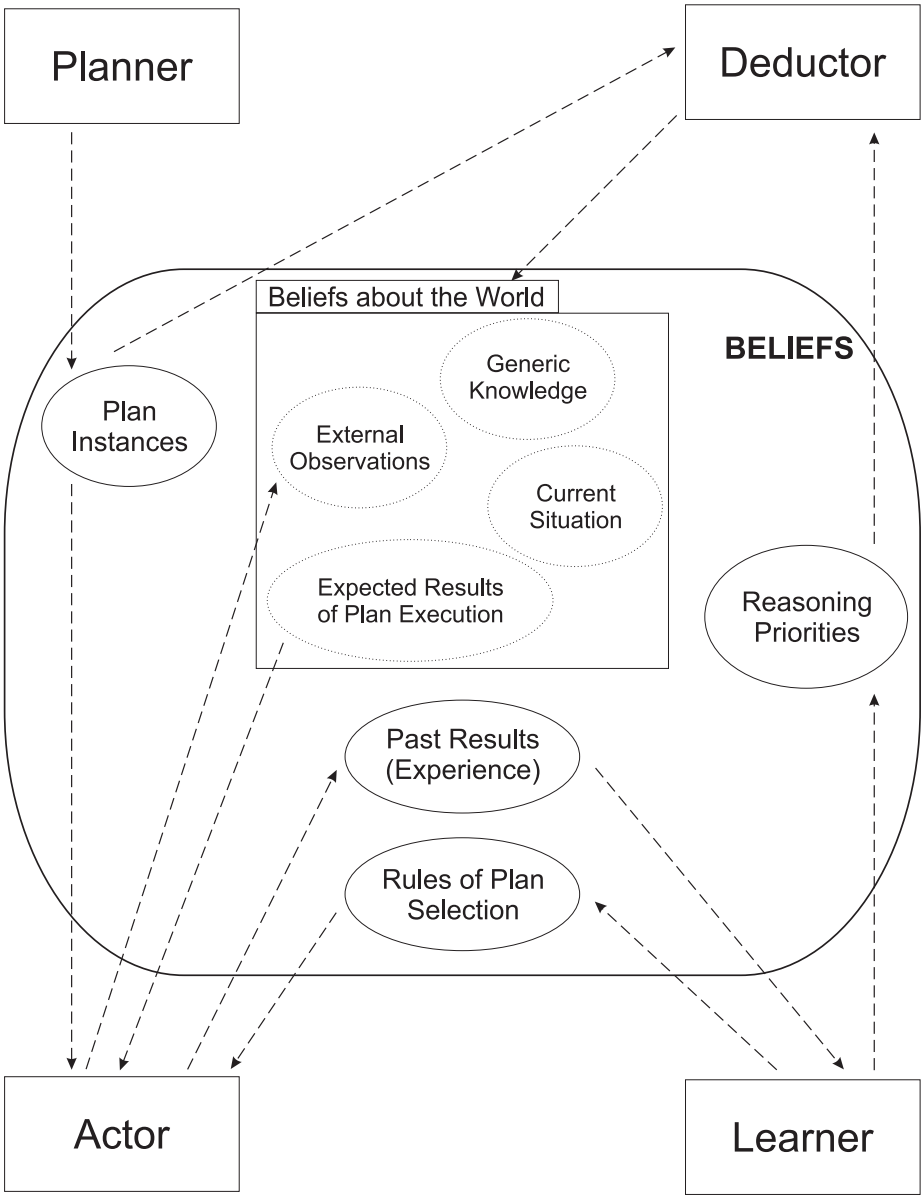

Figure 2.2: Information flow within the agent.

research is very important and often successful, it is our strong belief that neither of these subfields *alone* can give us truly intelligent, rational agents. Our architecture, which, to the best of our knowledge, is novel, may be one way to integrate them.

2.3 Knowledge Representation

Our agent's design combines a number of solutions coming from a number of subareas of Artificial Intelligence. There is a history of research being done on integration of the different approaches, and doing this has never been easy. While our architecture is by no means the ultimate answer to this integration problem, we believe that our setting is a rather promising one.

One of the main reasons for it being successful is our use of multiple conditional partial plans as the core knowledge representation formalism within the agent. They serve as a middle ground, a universal language which guarantees that each of the modules — even if they are built according to quite different principles — has the same understanding of basic concepts and limitations of the agent. Deductor reasons about different possible configurations of the environment, yet each of those situations corresponds to the state of the world after the agent executes a particular plan. Planner generates partial plans, but it is not forced to make perfect predictions of their results or to fully evaluate them, instead it is allowed to suggest a set of *potentially* interesting plans to the other modules. Actor performs the plan evaluation and selection, taking into account the plans themselves (as generated by Planner) and their expected consequences as deduced by Deductor. Finally, Learner induces the interesting facts about those plans, classifying them into various types, such as "useless", "interesting", "dangerous", etc.

The conditional plans we consider consist of a concatenation of classical and conditional actions, where each conditional action may be described (in a C-like notation) as:

$$(predicate\ ?\ plan_1 : plan_2),$$

meaning that $plan_1$ will be executed if $predicate$ holds, and $plan_2$ will be executed otherwise (both $plan_1$ and $plan_2$ may be conditional themselves). We consider the possibility of introducing a more complex structure of conditions (like *while* loops), but within the applications we investigate in this work simple conditionals do just fine.

This type of conditional actions introduces a high branching factor in the case of longer plans. This effect, however, is unavoidable at some level of consideration and will not be further discussed here. In the case of our research, this is largely mitigated by keeping the plans partial, and therefore short. This issue has received some attention in the works of other authors (see [GNT04] for extended bibliography).

For example, in the Wumpus domain, with an agent on square $a1$ (see Figure 2.3), one simple plan is "$a2$", meaning "go to $a2$". Another plan is "$a2,\ a3$", meaning "go to $a2$ and then go to $a3$". Both those plans are sequential. If the ultimate goal of an agent would be to reach square $a3$, than the first of those plans would be partial, while the second would be complete.

A conditional plan could be:

$$a2,\ (Smells(a2)\ ?\ a1\ :\ b2)$$

meaning "go to $a2$ and if it smells there go back to $a1$, otherwise go forward to $b2$". Since we will be discussing conditional plans in the Wumpus domain often throughout this thesis, from now on, we will introduce the following simplification. We will omit the "Smells" predicate, since it is the only one meaningful in this domain. Moreover, the argument of this predicate will always be the most recently visited square. Therefore, the above plan will be simply written as "$a2\ ?\ a1 : b2$".

a3	b3	c3
a2	b2	c2
a1	b1	c1

Clear	*Clear*	*Smells*
Clear	*Smells*	*Wumpus*
Player	*Clear*	*Smells*

Figure 2.3: Simple Wumpus board.

In the Wumpus domain it is difficult to find a good sequential plan which would be longer than one step. At the same time, finding a good conditional plan of length two or more is quite easy. In the experiments reported in this thesis, we consider plans of length one and two only.

Chapter 3

Deductor Module

3.1 Introduction

In a sense, Deductor forms the core of our agent, since it performs the logical inference and directly reasons about the knowledge that the agent possesses. In particular, it is the module that analyses not only the current state of the world, but also how it will change as a result of performing an action.

To this end, the agent uses a variant of Active Logic [EKM$^+$99], augmented with some ideas from Situation Calculus [Rei01]. This allows us to model the agent's reasoning as an ongoing process, thus explicitly putting resource limitations into the picture. This formalism is also well suited to talk *about* the agent's knowledge and to provide a wide range of introspection capabilities. Finally, the mechanisms for handling inconsistencies in the knowledge base allows us not to distinguish between knowledge and belief, instead assuming that *everything* our agent has deduced (or induced) can be challenged.

3.2 Situation Calculus Formalism

At the foundation of the language used by Deductor lies First Order Logic, augmented with Situation Calculus mechanisms for describing action and change. Within a given situation, knowledge is expressed using standard FOL syntax. We do not put any additional limitations on the expressiveness of the language as some mechanisms we later employ would invalidate the benefits of restricting ourselves to languages such as Horn clauses or description logics. A special predicate, $Knows$, describes the knowledge of the agent, e.g.,

$$Knows[\ Smells(a) \leftrightarrow \exists_x(\ Wumpus(x) \wedge Neighbour(a, x))\ ]$$

means: *the agent knows that it smells on exactly those squares that neighbour Wumpus's position*. The predicate $Knows$ may be nested, and while it is very useful, we employ it only in a fixed number of contexts (as explained below), so as to maintain reasoning at a sufficiently efficient level. We use a standard reification mechanism to put formulae as parameters of the $Knows$ predicate, as introduced in [FHVM95].

Reasoning within a given situation is not very interesting from our point of view, however. The whole point of Deductor is to be able to represent actions and changes within the environment. We base our approach on the Situation Calculus formalism described by Reiter in [Rei01]. Situation Calculus introduces two special predicates: $Holds(\mathbb{S}, \alpha)$ to denote that formula α holds in situation $\mathbb{S}$, and $Informs(\mathbb{A}, \alpha)$ to denote that action $\mathbb{A}$ provides information on whether the ground formula α holds. It also introduces a special function, $Result(\mathbb{S}, \mathbb{A})$, which returns the set of situations that may result from applying action $\mathbb{A}$ in situation $\mathbb{S}$.

In our approach we slightly modify the above, focusing on the agent's knowledge rather than representing the true state of the world. This allows us to use only one special predicate: $Knows$. A major concept in our formalism is a plan, which at this point can be thought of as a sequence of actions (although, as we have explained earlier, it can be more complex than that). In this context, the validity of a FOL formula depends not only on the actual situation, but also on the plan an agent is currently considering. In particular, an agent can reason about formulae of the kind:

$$Knows[\mathbb{S}, \mathbb{P}, \alpha],$$

where α is a FOL formula, $\mathbb{S}$ is a situation and $\mathbb{P}$ is a plan. Intuitively, it means: *the agent knows that after executing plan $\mathbb{P}$ in situation $\mathbb{S}$, formula α will hold*. It is important to observe that formula α itself may contain a nested predicate $Knows$, thus allowing the agent to reason about knowledge-producing actions. Such a method of double indexing allows us to express $Holds(\mathbb{S}, \alpha)$ as

$$Knows[\mathbb{S}, \emptyset, \alpha],$$

thus stating that formula α holds in situation $\mathbb{S}$ *and that the agent knows this*. Note that we are not at all interested in expressing facts the agent does not know in this way. Similarly, $Informs(\mathbb{A}, \alpha)$ can be expressed as:

$$Knows[\mathbb{S}, \mathbb{A}, Knows(\mathbb{S}^*, \emptyset, \alpha) \vee Knows(\mathbb{S}^*, \emptyset, \neg\alpha)]$$

at which point it is important to note that the equivalence is not complete. In our setting, the agent by default assumes that results of its actions depend on the current situation, while in the classical Situation Calculus, action results are by default situation-independent.

In the next section we show in more detail how the use of Active Logic allows our agent to perform introspection and take resource limitations into account.

3.3 Active Logic Formalism

Active Logic [EKM$^+$99] is a reasoning formalism which, unlike classical logic, concerns itself with the *process* of performing inferences, not just the final outcome (i.e. fixed-point) of the entailment (consequence) relation. In particular, instead of the classical notion of theoremhood, AL has so called i*-theorems*, i.e. formulae which can be proven *in* i *steps*. This allows our agent to reason about the *difficulty* of proving something, to retract knowledge it found inappropriate and to resolve contradictions in a meaningful way. It also makes the agent aware of the passage of time and of its own non-omniscience. An in-depth description of Active Logic, and especially its way of handling time, can be found in [PPT$^+$99].

In particular, each formula in AL is annotated with a time-step label (usually an integer) of when it was first derived. Moreover, Deductor keeps a record of the reasoning process and every application of an inference rule by incrementing this label. For example, the *modus ponens* inference rule looks like this:

$$\frac{i : \alpha, \alpha \Rightarrow \beta}{i+1 : \beta}$$

and means "if at step i formulae α and $\alpha \Rightarrow \beta$ are present in the belief set, then at step $i+1$ formula β will also be present." Moreover, there is a special inference rule:

$$\frac{i : Now(i)}{i+1 : Now(i+1)},$$

which allows an agent to refer to the current moment and to explicitly follow the passage of time, for example to conclude whether a deadline has been passed or not.

An additional feature available in Active Logic and important for this work is the *observation function*. This function delivers axioms that are valid from a specific point in time and is used to model the agent acquiring new knowledge from the environment. It is especially nice since it can easily describe changes that are not the result of performing any action, this way modelling *external* events. These two features allow us to overcome two important limitations present in the classical Situation Calculus.

Another important advantage of AL is its ability to handle inconsistency, which allows us not to distinguish between *knowledge* and *beliefs*. We assume that any part of agent's knowledge can be incorrect, and some of it may even be contradictory. We found this possibility to be extremely valuable in accommodating the results of inductive learning into the agent's knowledge base.

3.4 Agent's Introspection

As explained in Section 3.2, we have decided to augment Active Logic with basic concepts from Situation Calculus. In particular, since the agent needs to reason about a changing world and the effects of executing plans in various situations, we index formulae both with the current situation and with the plan being considered. Therefore, a typical formula our agent reasons about looks like this:

$$Knows[\,\mathbb{S}, \mathbb{P}, \forall_a Smells(a) \leftrightarrow \exists_x (\,Wumpus(x) \wedge Neighbour(a, x))\,],$$

and means "an agent knows that after executing plan $\mathbb{P}$ in situation $\mathbb{S}$, it will smell on exactly those squares which neighbour Wumpus's position". This particular formula is only mildly interesting, since it is true regardless of the chosen $\mathbb{S}$ and $\mathbb{P}$ (it is a universal law). If we denote the set of all situations by $\mathbb{S}^*$, the set of all plans by $\mathbb{P}^*$ and an empty plan by $\emptyset$, we can equivalently say[1] that:

$$Knows[\,\mathbb{S}^*, \mathbb{P}^*, \forall_a Smells(a) \leftrightarrow \exists_x (\,Wumpus(x) \wedge Neighbour(a, x))\,]$$

although, obviously, our agent itself is rarely explicitly aware of this for *all* infinitely many possible situations.

[1] Slightly abusing the notation

On the other hand, many — if not most — of the truly interesting formulae are true *only* for specific $\mathbb{S}$ and $\mathbb{P}$. This is especially true for those which are directly tied to the goals of the agent. For example:

$$Knows[\mathbb{S}, \mathbb{P}, \neg Wumpus(b2)]$$

which means "an agent knows that after executing plan $\mathbb{P}$ in situation $\mathbb{S}$, Wumpus will definitely not be on square $b2$" *does*, clearly, depend on $\mathbb{S}$.

Initially the agent is not able to state such a fact, but since its knowledge changes as it acts in the world it may, at some point, be justified. It is not necessarily immediately obvious that this particular formula still does not depend on $\mathbb{P}$, however. Nevertheless, it is clear that no really *new* knowledge can be obtained by simply *considering* some plan (without actually executing it).

If an agent $Knows[\mathbb{S}, \mathbb{P}, \neg Wumpus(b2)]$, then it must also be able to deduce $Knows[\mathbb{S}, \emptyset, \neg Wumpus(b2)]$, where $\emptyset$ stands for an empty plan. However, it does not necessarily need to be *aware* of the this fact right away, due to its lack of omniscience and limited reasoning resources. In fact, there are cases where the agent acts based on:

$$Knows[\mathbb{S}, \mathbb{P}, \neg Wumpus(b2)]$$

$$\neg Knows[\mathbb{S}, \emptyset, \neg Wumpus(b2)]$$

since it does not have the capacity to discover that $\neg Wumpus(b2)$ is also true. This is why the ability of Active Logic to handle inconsistencies is so important.

Finally, it is worth noting that since in our game the position of Wumpus's can never be changed, the actual validity of "$\neg Wumpus(b2)$" remains constant throughout the whole game episode. The only thing that changes is the agent's awareness of this fact. Therefore, if considerations related to some plan $\mathbb{P}$ ever lead an agent to (unconditionally) "$Knows[\mathbb{S}, \mathbb{P}, Wumpus(b2)]$", it can deduce that the $\mathbb{P}$ and $\mathbb{S}$ themselves are irrelevant and that this fact must hold for all possible plans and situations.

In contrast, an example of some really interesting formulae that can be deduced by the agent is:

$$Knows[\mathbb{S}, \emptyset, Knows[\mathbb{S}, \mathbb{P}, Wumpus(b3)] \vee Knows[\mathbb{S}, \mathbb{P}, Wumpus(c2)]]$$

which means "an agent knows that after executing plan $\mathbb{P}$ in situation $\mathbb{S}$, it will *either* know that Wumpus is on $b3$ or that Wumpus is on $c2$".

Which one of the "or" clauses will actually be true obviously depends on observations that the agent makes while acting. As an example of reasoning by cases and predicting action results, this is exactly the kind of knowledge we want the agent to infer — it *does* tell important things about the quality of the plan being considered. If all the agent knew before was:

$$Knows[\mathbb{S}, \emptyset, Wumpus(b3) \vee Wumpus(c2)]$$

then clearly executing $\mathbb{P}$ is useful — it will lead to the agent finally discovering the true position of the Wumpus. For a human expert, such $\mathbb{P}$ looks like a good plan.

The goal of our research is to enable *an agent* to reason about plans in exactly this way. It is our intuition, supported by experiments presented in Chapter 8, that it is possible to create a *domain independent* Actor module which would efficiently select good plans by learning from experience using formulae like the one above.

3.5 Reasoning

Finally, the representation language needs to be augmented with reasoning capabilities. This is done using a set of rather natural, yet not quite trivial, inference rules. For general purpose deduction, however, a simple *modus ponens* is quite sufficient. Using domain knowledge representing the rules of the Wumpus game, Deductor may conclude that

$$\begin{aligned} \forall_x\, Knows[\mathbb{S}, \mathbb{P}, \neg Smells(a) \wedge Neigh(a, x)\,] \quad &\leftrightarrow \\ Knows[Result(\mathbb{S}, \mathbb{P}), \emptyset, \neg Wumpus(x)\,] & \end{aligned}$$

i.e., that *if it does not smell at position a then the agent will know that there Wumpus is not on any of its neighbour squares*. If the agent currently is, or has been before, on square a, it may know that it does not smell there. This, in turn, may be used to create a useful plan of actions for the future, or to determine whether a particular action is a safe one.

The reasoning is also used to analyse plans generated by Planner and to determine their possible outcomes. Agent is mainly interested in knowing Wumpus's position, and, as explained in the previous section, different plans will have different effects on the agent's knowledge about it.

Another thing is to discover whether a plan is safe or not. For example, if the agent is able to deduce that

$$Knows[\mathbb{S}, \mathbb{P}, \neg dead(agent)]$$

then plan $\mathbb{P}$ appears to be an interesting one. If $\neg dead(agent)$ cannot be deduced, than it is a sign that a better plan may be needed.

One of the reasons for which we have chosen a symbolic representation of plans, as opposed to a policy (an assignment of a value to each state–action pair) is that we intend to deal with other types of goals than just reachability ones. For a discussion of possibilities and rationalisation of why such goals are interesting, see for example [BCPT03], where authors present a solution to planning using goals described in Computational Tree Logic. This formalism allows to express goals of the kind "value *a* will never be changed", "*a* will be restored to its original value" or "value of *a* after time *t* will always be *b*" etc.

Furthermore, one of our ideas is to extend the solution presented in this thesis to the case of multi-agent cooperative planning, where benefits of symbolic plan representation are even more apparent.

In order to make plan evaluation more meaningful, we allow plans not only to be simple (sequential) but also *conditional*, i.e. to contain branches where actions depend on observations made during acting. We believe that such conditional plans will, in many domains, be much easier to classify as either good or bad ones, since they contain more *generic* knowledge and have a greatly expanded applicability. A potential problem lies in making sure that, during execution, the agent has enough information to choose an appropriate branch of the conditional plan. In our current implementation this problem is solved by Planner, however, it is conceivable that Deductor might "validate" each plan in this sense as well.

Chapter 4

Planner Module

4.1 Introduction

As we stated earlier, our agent's reasoning is based on conditional, partial plans, and it is the Planner module that is responsible for their generation. Taking into account Deductor's knowledge about the current situation, Planner comes up with a number of plans that are potential ways to achieve the goals of our agent. The working assumption in our framework and throughout this whole thesis is that the domain is too complex for an agent to be able to develop an optimal, complete plan for solving the problem at hand. As we have explained in the Introduction, this quality is very common among typical, practically useful domains.

4.2 Plan Generation

One of the biggest problems in generating plans for rational, situated agents is the combinatorial explosion of the possible action outcomes. As all interesting problems in Artificial Intelligence, creating an optimal plan is NP-complete. Even in a so called "classical setting", the complexity of the planning problem is exponential in the number of possible states of the world. Nevertheless, there exist a number of very efficient algorithms for solving classical planning problems. They are, however, based on a number of quite restricting assumptions: the agent needs to have perfect knowledge about the environment, all actions need to have deterministic outcomes and only agent's actions may influence the world (no external events).

On the other hand, if the agent's knowledge about the world is not complete,

most actions have more than one possible outcome. This makes the planning process even more computationally expensive. For nondeterministic and partially unknown domains, its complexity is exponential in the number of agent's belief states. As of now, there are no sufficiently efficient, domain independent planning algorithms which are able to handle such complex settings.

For this reason, all domain-independent efficient planning algorithms make simplifying assumptions about the external world and about the agent's capabilities, in order to achieve a satisfactory performance. If those assumptions are not satisfied, however, the results of the planning process may not be applicable to real environments. This is a tradeoff that researchers need to be aware of, and for which no truly satisfactory solution has been found up to now.

There is also an active field of research concerning ways to incorporate domain-specific knowledge into planning (see for example [BFT07]). A number of successful applications has been developed based on such approaches, and they are certainly one of the possibilities of dealing with complex environments. Within our setting, however, we use a different approach, assuming that the agent's experience can be seen as a kind of "automatically obtained domain specific knowledge".

A crucial feature of our architecture is the fact that other modules are expected to analyse conditional partial plans coming from Planner and to improve them. In this way we are able to use the necessary simplifications in order to make plan generation sufficiently efficient, while at the same time we have ways to deal with the unavoidable discrepancies between the real world and Planner's simplified view of it. In particular, since the generated plans are analysed further by other parts of the agent, the results of the planning process are allowed to be approximate.

Within our architecture, we allow for two distinctive ways in which plans can be analysed further: using deduction (as we have explained in Section 3.5) and using induction (as we will explain in Section 6.2). Those two approaches complement each other very well and combining them often produces good results, and we show one example in Chapter 8.

In our setting, Planner is allowed to generate a number of potentially interesting conditional partial plans, without having to commit to a single one. The most difficult issue, in practice, is often how to determine which of the several possibilities is *the best* one. As in the case of many difficult problems, there is a law of diminishing returns complicating matters: it is relatively easy to improve initially and to find a number of "quite good" solutions, but as one approaches the ideal, improvements become more and more difficult.

4.3 Relaxation

In the design of our agent we focus on one common setting where it is beneficial to be able to generate a number of plans without deciding that one of them is better than the others. This is the case of an efficient (often classical) planner being used via application of the so called *relaxation* technique. In such setting, it is very useful to be able to evaluate a number of plans only *after* they are generated.

Classical planners are often very efficient reasoners, but they achieve this efficiency by strongly enforcing a number of limitations on how domain is modelled — often very constraining ones. As mentioned above, a typical assumption is that the agent has complete knowledge about the state of its environment. In the domains we are interested in, for example in the game of Wumpus, as well as in many realistic situations, such an assumption is completely unfounded. Therefore, it is impossible to directly take advantage of those efficient algorithms.

They can be used indirectly, however. Relaxation consists of, basically, making the problem simpler by removing some of its complexity (see [GNT04] for extended bibliography). For example, a Wumpus problem could be relaxed by allowing the agent to directly sense the position of the monster. A game in which the agent is allowed to perform such "super-sensing" action is obviously much easier than "real" Wumpus, but they still share a number of similarities. And while a plan developed in this easier domain is not directly applicable in a normal game, some insight may be extracted from it.

In particular, it is easy to devise a large number of different ways to relax any given problem. At the same time, if those relaxations are done in the right, systematic way, a number of them will result in different plans. This way a large number of potentially interesting solutions can be obtained by using a relatively low amount of resources. If they can be combined in some way, or if the agent is able to make smart decisions about which of these plans is the best one, that approach can offer very good performance.

For example, we can imagine an agent playing a simplified Wumpus game where it is able to decide what observation it "wants" to obtain. Normally, when the agent enters some new square, it has no way of knowing whether it will smell there or not, and no way of influencing it either. But one possible relaxation algorithm is to allow an agent to treat all observations as actions and to "choose" how it wants the world to look like. In this setting, the agent starting at $a1$ may decide to move to $a2$ and may also *decide* that it wants it to smell there.

An important observation is that plans resulting from solving relaxed problems tend to be relatively accurate at the beginning, but divert from reality rapidly as their length increases. This is directly caused by the fact that the inaccuracies in modelling the world are kept as small as possible, but they show a tendency to accumulate as more and more "unreal" actions are used. In general, this is a serious problem for agents trying to utilise various kinds of relaxation approaches.

Our architecture, however, is designed in such a way that the agent can capitalise on the initial high quality of plans and avoid the trap of uselessness of long plans. This is due to the *partial* nature of plans it employs — it simply takes into account *only* the beginning of a relaxed plan and disregards the ending. For example, in our current implementation, we only consider plans of length one and two.

Similarly, many relaxation techniques naturally allow an agent to create conditional plans. By their very design, most relaxations consist of the agent making "extra" choices during planning, by influencing features that are normally out of its control. Each such "unnatural" choice suggests itself like a good place for a conditional branch in an un-relaxed plan. If a relaxed plan can be seen as a plan which "works" if a number of events over which the agent has no control fall in a particular way, it is only natural to ask what happens if they do not.

For example, using the relaxation we introduced above, we are "pretending" that the agent is allowed to determine the result of a particular observation. In such setting, a good plan might consist of five actions: "go to $a2$", "observe $Smells(a2)$", "go to $a1$", "go to $b1$", "observe $Smells(b1)$". This suffices to win the game, since after such a sequence of events the Wumpus must be on $b2$. It is also clear that the relaxed problem is, indeed, easier than the original one: in real Wumpus, it is not quite as easy to win.

However, this is obviously not the only plan which allows the agent to win the game. Another one would be: "go to $a2$", "observe $\neg Smells(a2)$", "go to $b2$", "observe $\neg Smells(b2)$", "go to $b3$", "observe $\neg Smells(b3)$". It is obvious that Wumpus must be on $c1$, then.

It can easily be seen that those two plans share the initial action, and then have different outcomes of the first observation. There exist a perfectly natural way to combine them into a single, conditional plan — which will branch on exactly this first observation. The result would be: "go to $a2$", IF "$Smells(a2)$" THEN "go to $a1$", "go to $b1$" and "observe $Smells(b1)$" ELSE "go to $b2$", "observe $\neg Smells(b2)$", "go to $b3$" and "observe $\neg Smells(b3)$". This plan is

still not necessarily valid for the "real" game of Wumpus, but it is sufficiently close to a good one (at least near its beginning) that it should illustrate our idea well.

Basing on that, it is equally easy to make the plan partial by cutting it on the first relaxed action. In our case, it would result in "go to $a2$ if it smells there go to $a1$ and to $b1$, otherwise go to $b2$" being generated. And while this plan is not yet complete (there is a chance it will not smell on $b1$ nor on $b2$), it is definitely a reasonable plan. And it is perfectly valid for the "real" Wumpus domain, with no trace of its relaxation heritage left.

Obviously, there is no guarantee that this plan is actually any good, since it could lead the agent to a dead end — in a relaxed domain, there was a solution, but it is at best a weak indication that in real domain this plan may be usable.

4.4 Plan Evaluation

As we explained in the previous section, our agent creates conditional, partial plans. These plans are partial because limited resources do not allow our agent to consider all the possibilities and come up with a good conformant plan. In fact, we would prefer to be able to generate conformant plans, and the only reason for plans being partial is that they are the best we can create.

On the other hand, the reason we have decided to use conditional plans, instead of limiting ourselves to sequential plans, is plan evaluation. We have shown how conditional plans can be created, and that it is, in fact, often easier than generation of sequential plans, but this is a secondary issue.

The primary reason for plans being conditional is that we intend the agent to learn that some of them are generally good and some of them are generally bad. By the virtue of being conditional, the plans remain concise but also have significantly broader applicability. The language of conditional plans is strictly more expressive, and this allows Planner to generate plans which are not confined to a single situation.

In particular, one of the major contributions of this work is the idea to inductively generate rules for deciding when a plan is a good one for executing in a particular situation and when it is a bad one. The agent uses Inductive Logic Programming in order to achieve that.

From this point if view, sequential plans can rarely be universally good, unless they are very short. For example, if we consider a plan "go to $a2$ then go to $a3$", it is obvious that it is only good under some rather strict constraints: if the agent knows that Wumpus is neither on $a2$ nor on $a3$. In particular, the

agent does not have this knowledge at the beginning of any game episode.

On the other hand, if we consider the, only slightly more complex, conditional plan "go to $a2$ and if it does not smell there go to $a3$", it is obvious that it is much more general. In fact, it is quite a good plan to execute at the beginning of a game episode.

We have based our experiments, reported in Chapter 8, on the idea of learning to distinguish "good" and "bad" conditional partial plans. This is only possible in a setting where Planner generates conditional partial plans and the agent is allowed to make use of past experience in order to determine a hypothesis as to what kind of plans are successful and which are not. The results of those experiments are very encouraging and they show that the framework we have designed is well-founded and promises to work well.

In our current implementation used in those experiments, we have decided to focus on the interactions between learning and deduction, so both Planner and Actor have been significantly simplified. Our planner is a simple one, it does not use any heuristics and simply creates a fixed set of plans only (all possible plans for the Wumpus domain, and an arbitrary set of "interesting" plans for Chess).

This is, however, only a simplification made in order to make the experimental setting more accessible. The agent's architecture is designed in such a way that it is very easy to hook up an existing planner in order to efficiently create the "reasonable" plans.

Chapter 5

Actor Module

5.1 Introduction

Actor is the least homogeneous module in the whole architecture, since it can be seen as a controller of the agent as a whole. Basically, its responsibilities can be divided into three broad classes.

Firstly, Actor supervises the deliberation process in order to ensure that the agent stays responsive in a dynamic environment and that the right balance between reasoning and acting is reached. Secondly, it selects the best plan to be executed. And thirdly, it continuously observes the environment and allows the agent to react to interesting events taking place there.

5.2 Reasoning

Actor oversees the deduction process to guarantee that the agent's limited resources are utilised to the highest degree possible. To this end it analyses the progress of reasoning and makes decisions as to when should it be interrupted. There are three major types of situations when a decision to give up on further thinking is justified.

First situation occurs when Actor notices a particularly interesting plan and decides there is no point in deliberating further. Definitely the most common and obvious condition for that is discovery of a complete plan. Even though Planner aims at generating partial ones, at some point the agent will be sufficiently close to achieving its goals that a complete solution can be found. Once it is, and Deductor confirms that ultimate goal of the agent can be reached directly, there is no point in continuing deliberation and agent should immediately

begin to execute this plan.

Second situation is probably more common, even if less satisfying. Namely, when the agent has spent enough time reasoning and will come to no further *interesting* conclusions, Actor needs to break the deliberation and try acting instead. It is crucial to note that this requires some measure of evaluating the progress of reasoning not based on purely syntactical measures such as number of formulae known, but on determining whether newly obtained knowledge is useful and relevant. For example, if an agent knows that 0 is a number and any successor of a number is the number itself, then it can easily continue to deduce infinitely many "new" formulae. However, most of them are, in themselves, completely useless and they will not be important for determining the correct course of action.

Finally, it may happen that the agent is pleased with the deliberation process, it has not yet found any particularly exquisite plan, but it still needs to start acting due to a deadline approaching. It is common that external events happen at predictable intervals and if the agent needs to be able to respond to such event, it may have to interrupt its own reasoning in a particular moment, regardless of how close it is to a breakthrough.

In general, however, it is impossible to determine whether the deliberation process is "progressing" within reasonable resource limitations (this problem is sufficiently difficult in the case of omniscient agents). However, within our architecture, we can take advantage of the learning mechanisms to help in this regard. More testing is needed, but based on the experimental results presented in Section 8.3, learning about conditional partial plans seems to be promising direction of research.

Another interesting idea is to allow Actor to "pause" reasoning and perform any observations Deductor requires to continue. In a number of situations it happens that some decision cannot be made without some crucial piece of information. In those cases it is counterproductive to continue deliberation, and it is much better to perform whatever actions are necessary to obtain this information. We have not implemented this functionality yet, but Actor has the means to detect such situations and to react to them.

For example, if an agent in RoboCup domain intends to kick the ball but does not know the distance to the goal, it may attempt to plan what the exact actuator gain should be used. Such agent might begin creating plans of the form "detect distance, if distance is 0.1 then set gain 1, else if distance is 0.2 then set gain 2 else if distance is 0.3 then set gain 3, ..." It is clearly more beneficial to perform the observation immediately rather than to waste time creating such a

huge conditional plan.

In addition to interrupting deliberation completely, Actor also guides the reasoning process by making it focus on the plans most likely to be useful. Based on plan evaluation capabilities as discussed in the next section, it is able to prioritise deduction by making it focus more effort on plans looking more promising. It utilises Learner and mechanisms discussed in Section 6.2 to do that.

In our current implementation and in experiments reported in Chapter 8, we have decided to focus on the interactions between learning and deduction, so both Planner and Actor have been significantly simplified. Actor relies on the incompleteness of the Deductor and on the fact that the inference is guaranteed to always terminate. It simply lets Deductor infer everything it can about each of the available plans and only afterwards does it choose the best one to execute.

5.3 Plan Selection

Once a decision to begin acting is made, Actor evaluates existing plans and executes the best one of them. This *evaluation process* is one of the most important aspects of the whole agent architecture. On the one hand, as explained earlier, it is not possible, in general, to determine with full confidence which of the available conditional partial plans is the optimal one. At the same time, Actor's choice of which plan to execute is the crucial element which directly influences the agent's ability to reach its goals.

For example, in the Wumpus domain, if the agent keeps executing the wrong plans, it can continue walking around the board in circles and never find the monster. Without domain-specific knowledge regarding the correct strategy for playing the game, partial plans are not guaranteed to move the agent closer to winning. Moreover, there is a number of domains that are not "recoverable", i.e. an agent may end up in a situation which is not fatal by itself, but from which winning is no longer possible. Many interesting, realistic domains are sufficiently complex to have that property. For example, even our simple Chess domain is one.

The decision made by Actor is partially based on deductive results obtained by Deductor about each candidate plan (see Section 3.5), and partially based on past experience and generalisations of it developed by Learner (see Section 6.7). In our approach, it is the learning process that makes the evaluation of plans feasible.

In the beginning, the choice of plan to execute is made at random (but

heuristics may be used if any are available in the domain-specific knowledge). After executing this random plan, a new situation is entered into and new observations are obtained. Planner creates another set of plans, taking into account the newly acquired knowledge about the world, Deductor analyses them, and Actor selects and executes one.

After this is done for a number of game episodes, however, the agent has enough experience to use the Learner module to generalise its past results and to induce some hypothesis as to which plans are good and which are bad. This part has been our major focus in this thesis, and the details are discussed further in Chapter 6.

Winning the game also provides a possibility to (re)construct a conformant (or, at least, more general) plan out of the partial ones used in the past, both in this episode and in previous ones. If such a plan can be found, it may be subsequently used to immediately solve any problem instance for which it is applicable.

Summarising, the interactions between Actor and Learner are the most important issue we report on in our experimental results in Chapter 8.

5.4 Execution

Finally, the Actor module is responsible for interactions of the agent with the external world, performing the role of the reactive part of the system. In the most typical scenario, Actor continuously monitors the sensor input, analyses it and transforms it into symbolic representation whenever needed. In Active Logic, there exists a special provision for that, called *observation function*. It is important that such newly observed formulae have exactly the same status as domain axioms, but different temporal extent.

In general, the task of turning complex sensor input into symbolic representation suitable for deductive reasoning is far from trivial. A number of researchers work actively on this topic (see, for example, [CS01]). Still, this is not a focus of this thesis and for our work we simply assume that an existing solution can be used.

Newly observed facts can then be used by the Deductor, allowing an agent to respond to the changes in the world. In particular, Deductor is well equipped to handle the common case of new information contradicting (possibly obsoleting) previously available one. Active Logic allows it to resolve inconsistencies in the agent's knowledge base and to make informed decisions regarding which source of information is more trustworthy (including common sense notions

like “believe the most recently acquired observations”).

Nevertheless, it is also possible for Actor to react more “violently” if the need arises. In some situations, it is not enough to rely on deductive process to be sufficient. The most typical case is when some physical actions need to be performed immediately. A classical example would be execution of obstacle avoidance behaviour if the agent is about to hit an obstacle. Typically, the deductive process is not going to react sufficiently quickly and, in dynamic environments, an agent needs to have a repertoire of “ready to use” responses to external events. It is also important that both Learner and Deductor cooperate in creating this repertoire.

In other words, the intention for Actor is to acquire generalised knowledge of the domain, which can be used to guide the agent in more promising directions.

5.5 Environment Interaction

One of the main contributions of our research lies in the “consciousness” of interactions between an agent and its environment, conducted in such a way as to maximise the knowledge that can be obtained. In particular, the agent is facing, at all times, the exploration versus exploitation dilemma, i.e., it both needs to gather new knowledge *and* to win the current game episode.

In order to facilitate that, our agent requires an ability to both act in the world and to observe it. Finally, it needs to reason about its own knowledge and how it will (or *can*) change in response to various events taking place in the environment. In different domains and applications various models of interactions with the world are possible. In this section we will describe how they can influence the agent.

The most unrestrictive case is a simulator, where an agent has complete control over the (training) environment. It can setup an arbitrary situation, execute some actions and observe the results. Such a scenario is common in, for example, physical modelling, where it is often much easier to simulate things than to predict their behaviour and interactions.

If an agent’s freedom is slightly more restricted, it is possible that it is not allowed to freely change the environment, but can “try out” several plans in a given situation. For example, the agent may provide a set of plans and receive an outcome for each of them. This model is also suitable for agents that do not have perfect knowledge of the world, as the “replay” capability does not assume that *the agent* is able to fully reconstruct the situation or that it knows the state

of the world completely.

In most applications, however, an agent is only able to influence its own actions and has no control whatsoever over the rest of the world. This is also the most suitable model for an *autonomous* physical agent. In such a case, the environment will irreversibly move into the subsequent state upon each agent's action (or any other event), leaving it no option but to adapt. It may still be interesting, in some situations, to substitute acting for reasoning, but the agent needs to be aware that once acted upon, the current situation will be gone, possibly forever.

Finally, we can imagine a physical agent situated in a *dangerous* environment, where it is not even plausible for it to freely choose its actions — it first needs to assert that an action is reasonably safe. In this case, unlike in the previous ones, a significant amount of reasoning *needs* to be performed before every experiment.

As an orthogonal issue, sometimes it is feasible for an agent to execute an action, observe the results, reason about them and figure out the next action to perform. But in many applications the "value" of time varies significantly. There are situations where an agent may freely spend its time meditating, and there are situations where decisions must be made quickly. For example, in the RoboCup robotic soccer domain, when the ball is in possession of a friendly player, the agent just needs to position itself in a good way for a possible pass — a task which is not too demanding and leaves the agent free to ponder more "philosophical" issues. On the other hand, when the ball is rolling in the agent's direction, time is of essence and an agent should have plans ready for several of the most plausible action outcomes.

Chapter 6

Learner Module

6.1 Introduction

The goal of the learning module is to take advantage of the experience the agent gathers throughout its lifetime. After all, one of the defining traits of intelligence is the ability to improve its own performance and to avoid repeating old mistakes.

The Learner module is intended to improve our agent's performance as a whole. Firstly, it provides Actor with knowledge necessary to choose the best plan among those suggested by Planner. Moreover, it induces rules for deciding when it is time to stop deliberation since no new interesting insights are likely to be obtained. Learner also provides Deductor with guidelines about plans that are most likely to lead to good results (and thus should receive priority when reasoning) and about plans that are likely to be useless (and should thus be ignored by Deductor).

6.2 Plan Selection

In this thesis we mainly investigate Learner from Actor's perspective, since using the Inductive Logic Programming framework to evaluate the quality of partial plans is — to the best of our knowledge — a novel idea. Nevertheless, the design of the Learner module as well as the agent architecture around it is well suited for more than just that. For example, it can be used to improve domain knowledge and to identify interesting reasoning directions.

As we mentioned earlier, our agent is expected to live through a large number of problem-solving episodes in a single domain. This is not an assumption

particularly difficult to fulfil in practice, as it is very atypical for an agent to solve some problem only once and to face a completely different task the next day. For example, if the agent is an autonomous museum guide, automatic taxi driver, office helper robot designed to bring coffee to hard working researchers or any similar kind of device, the tasks it faces are quite repeatable, even if never completely identical.

At the same time, the ability to exploit its past experience is one of the greatest assets of our agent. For example, upon finishing each Wumpus game episode, all the interesting events (actions, observations, and finally the ultimate result: whether the game was won or lost) are fed into the learning module. Learner attempts to generalise this information and to provide guidelines for Actor and Deductor on how to achieve a better performance next time.

The first task for our learning module is to help Actor to choose the best one among the plans being considered by Deductor for execution. As explained in previous chapters, Planner develops a number of conditional partial plans and then Deductor analyses each one of them (see Section 3.5 for details). Some of those plans are better than others, but it cannot be determined exactly and with full confidence unless those plans extend all the way to the ultimate goal of the agent (for example, to the terminal state of the game). For the kinds of domains we are interested in (such as those mentioned in the Introduction) doing that is infeasible — the agent's computational resources are not enough to *completely* solve these problems.

Therefore, the agent needs some heuristic method, preferably adapted to a particular domain, for evaluating the quality of partial plans and comparing them. After all, Actor must, at some point, choose exactly one plan for immediate execution. There is an active research in planning that focuses on exactly this problem, namely how to automatically generate suitable heuristics. Still, the most important, if not the only, generic result achieved is that all solutions are domain-dependent and what works very well in one situation may be quite detrimental in another. Using Machine Learning techniques [Mit97] is one way of dealing with this issue.

To this end, Learner inductively discovers rules for evaluating plans and for deciding, based on the agent's successes and failures from the past, which ones are most likely to lead to the goal. Doing so using deductive reasoning is often computationally very expensive, and it is a wise idea for an agent to avoid it whenever possible. Based on experience and on deductive reasoning performed by the Deductor module, the agent can analyse how the world (and the agent's knowledge about it) will change after executing a particular plan.

It is then possible to learn rules for determining the class of plans which have been successful in the past, and to use that to choose the one to be executed next.

6.3 Inductive Logic Programming

There exists a large number of different approaches based on the idea of exploiting past experience, under the common name of "Machine Learning". Some of them attempt to build models of the process in question, using various formalisms, and later exploit it (for example, approaches based on the Markov Decision Processes). Others focus more on a procedural knowledge, attempting to directly determine the correct course of action, often in the form of a *policy* (for example, Reinforcement Learning). Others concern themselves only with classification of entities in the world, without directly associating such decision classes with the agent's actions (various kinds of Data Mining).

In our research we have decided to use the approach called Inductive Logic Programming, since it has a number of very attractive advantages important in our setting.

The major requirement is that training knowledge needs to be expressed in a very rich language — in particular, simple attribute-value pairs are not sufficient to describe training examples. It is crucial for a learning system to have access to the complete knowledge base of our agent. And while there exist a number of other formal languages, the full expressiveness of FOL is still needed to capture the complex dependencies between different elements of the world. Inductive Logic Programming is one of the few Machine Learning techniques that can support such complex descriptions of training examples.

Moreover, very often there is quite a bit of knowledge that domain experts can provide. While our aim is to have a solution that does not *require* such experts, we believe that taking advantage of whatever information they can provide is very important. At the same time, we intend to put as few requirements on this knowledge as possible — it is all too common for Artificial Intelligence approaches to put constraints on domain knowledge, which basically amounts to "perfect knowledge or no knowledge". In contrast, Inductive Logic Programming fits our needs quite well — it uses background knowledge when it is available, but it can also solve problems when it is not. At the same time, the full expressiveness of FOL is available, and the only formal constraint is that provided knowledge cannot be inconsistent.

The main problem is that most of the work on ILP (and to a lesser extent

on symbolic Machine Learning in general), has been dealing almost exclusively with the problem of *classification*. In contrast, our setting requires *evaluation* or *ranking* instead. There is no predefined set of classes into which plans should be assigned — what our agent needs is a way to choose the *best* one of them. It must do so regardless of how many are available or how good — on an absolute scale — they are.

Still, in order to be able to take advantage of the vast amount of research done in the Inductive Logic Programming framework, we try, at least initially, to recast our problem as one of classification. To this end, instead of trying to determine which plan is the "best" one, we simply attempt to divide the set of all plans into two classes: plans that are "obviously useless" and plans that are "sensible".

Even such a simplified setting can be useful, since if Actor can spot those "obviously bad" plans early, it can instruct Deductor not to waste time deliberating about them. Clearly, these plans should never be executed, and since the fact that they're bad is so obvious, it is quite likely that the agent will determine that anyway. Nevertheless, it is very typical that a lot of effort is wasted before that happens. Generalising agent's past experience can be a great way to minimise this wasteful effort.

In our example game of Wumpus, we have decided to consider as "bad" those plans that are "dangerous", i.e. those that can lead to the agent being eaten by the monster. Clearly some plans — namely those that in agent's experience *did* lead to death — are bad ones. It is not common, however, for the complete and operational definition of bad plan to directly follow from the available domain knowledge. It is even less likely that an agent will be able to explicitly deduce it and to use it effortlessly.

It is important to understand that this setting is only a beginning. After all, in many situations a more "proactive" approach than simply *not-losing* is required. For example, an agent that moves in circles, without exploring the world, clearly does not get eaten by the Wumpus — but it never wins the game, either. Hardly is it worth being called "intelligent". When going further, one very promising idea is to explore the epistemic qualities of plans: an agent should pursue those plans which provide it with the most important knowledge about the world. Inductive Logic Programming appears to be useful for such a setting as well, but initially we focus on a simpler issue.

Another way of expressing the distinction between good and bad partial plans, and one we feel can give satisfactory results, is discovering relevant subgoals and landmarks, as [HPS04]. The ability to divide complex problems into

easier components and to solve them separately is a very powerful technique. Again, Machine Learning can be quite successful in this.

6.4 Training Knowledge

There is a large number of features that can be used to distinguish between good and bad plans. And with a sufficiently rich history of past game episodes, it is possible to learn this distinction. In fact, there are important theoretical results which prove that, given a sufficiently large and sufficiently random training data set, the probability that a properly designed learning procedure will generate a bad hypothesis (i.e. one making a large error) is arbitrarily small.

How such approaches work in practice, however, is far from obvious. There is a number of reasons why such theoretical results are not necessarily fully applicable in reality. In particular, a number of constraints faced by situated agents make most theoretical analyses rough simplifications at best. Therefore, it is important to investigate in more detail the actual quality of learning within our framework.

In the simplest case the agent can start with Actor randomly choosing plans for execution. After a couple of games — some of which will be won but, most likely, many will be lost — it should have enough experience to learn some useful rules. How complex those rules would be, how intuitive and whether a particular learning algorithm will discover them (as always, within the agent's limited resources) is something that needs to be experimentally verified, and we have done that in Chapter 8.

We start with the idea that Learner only attempts to distinguish "dangerous" plans, i.e. ones that can lead the agent to failure at its task. Looking at our example domains, the most natural definition of a dangerous plan in the Wumpus domain would be "a plan which can lead to the agent's immediate death". Similarly, in a Chess domain, a dangerous plan would the be one which can lead to losing the rook. This is obviously not enough to achieve the agent's true intelligence nor to ensure that it will be successful in achieving its goals in general. Nevertheless, there is a large class of domains which are "recoverable" in the sense that (as long as agent is not dead) it can still win, regardless of current position. In fact, Wumpus domain belongs to this class, while Chess does not.

However, it is worth noting that if the Wumpus is allowed to move, there exist plans which do not lead to agent's death, but which nevertheless can make the game unwinnable — for example, if an agent gets stuck in a corner with Wumpus blocking its way out. It may be difficult for an agent to notice and

learn that the mistake has been made in the previous step, not in the one when the agent was killed. This issue is most extensively discussed in the area of Reinforcement Learning, and it is closely related to the problem of delayed rewards and credit assignments. An extensive bibliography is available, for example, in [SB98].

Nevertheless, in the rest of this section we restrict ourselves to dividing plans into two classes: those that can lead to agent's death and those that cannot. Each partial plan executed at some previous game can be seen as a single training example. The first issue we need to deal with is which example belongs to which class.

It is easy to notice that some plans — namely those that in agent's experience *do* lead to losing the game — are definitely examples of bad plans. However, not every plan which does not cause the agent to die is, indeed, a *good* plan. What more, not every plan that leads to *winning* a game is a good one. An agent executing a dangerous plan might have just gotten lucky, if in a particular episode Wumpus was in favourable position. From this it becomes clear that the notion of *positive* and *negative* examples, as used in ILP algorithms, is not quite appropriate for what we would like to express within our framework.

In the experiments reported on in Chapter 8, we assume that the agent has perfect knowledge about which plans (training examples) are potentially bad. This is a fully justified assumption for the Chess domain, where the opponent does not make trivial mistakes and whenever it is possible for him to capture the rook, he will do so — therefore if a plan is bad, it *will* cause the agent to lose the game.

In Wumpus, however, the distinction is not so clear — it is possible for the agent to get lucky and not die even though it executes a dangerous plan, simply because the beast is in a favourable position. Still, in the simplest case, within our current experimental setting, this problem can be avoided. Namely, we assume that Deductor, even if incomplete, has perfect knowledge about the rules of the game and it is powerful enough to eventually discover whether there is any possibility of the agent dying due to executing a particular plan in a given situation or not.

By knowing all possible consequences of the execution of each plan, the agent can deduce (for some plans $\mathbb{P}$) a fact "$Knows[\mathbb{S}, \mathbb{P}, \neg die(agent)]$", and for others a fact $Knows[\mathbb{S}, \mathbb{P}, die(agent)]$. Since we intend to learn a predicate "$badPlan(\mathbb{P})$", we can define as positive examples those plans which lead — or can be proven to *possibly* lead — to defeat. Those are exactly the plans for which $Knows[\mathbb{S}, \mathbb{P}, die(agent)]$ can be deduced for at least one conditional

branch. On the other hand, plans which can be proven to *never* cause defeat are negative examples. In Chapter 8 we show that, under reasonable conditions, it is possible for Learner to inductively learn the correct definition of the $badPlan$ predicate.

This is not the ideal setup, however. Requiring Deductor to reason so deeply about every potential example for Learner is very demanding computationally, and for the results of those inferences to necessarily be *correct*, is very limiting. We would prefer to be able to label training examples using experience alone: in game episodes which ended up with the agent getting eaten, (some of) the executed plans were definitely bad. Equivalently, in game episodes which ended up with the agent winning, (some of) the executed plans were definitely good. In such a setting, however, there is no way to ensure that no mistakes happen when labelling training examples. And even though PROGOL caters for the possibility of noisy data, we have found its features in this area rather insufficient for this particular application.

There is also a third class of examples, i.e. plans for which the above cannot be proven: neither "$Knows[\mathbb{S}, \mathbb{P}, \neg die(agent)]$", nor $Knows[\mathbb{S}, \mathbb{P}, die(agent)]$, at least within the limited resources of our agent. We are working on how to most effectively use such examples in learning, but this is a secondary issue for now, since with our current implementation of Deductor this class is empty.

6.5 Learning Algorithm

In this work we use the Inductive Logic Programming algorithm called PROGOL [Mug95], since it is one of the best known ones and its author has provided a fully-functional, publicly available implementation. At this stage of our research we are not aiming for top performance, but rather to convincingly present our ideas, and to this end such a popular and well-understood algorithm is perfect.

PROGOL is based on the idea of *inverse entailment* and it employs a covering approach similar to the one used by FOIL, in order to generate a hypothesis consisting of a set of clauses which cover all positive examples and do not cover any negative ones. An important feature of PROGOL are *mode declarations*, where the user specifies which predicates can be used in the hypothesis being learned, as well as their arity and argument types.

The standard version is presented in [Mug95] and can be described here, in a somewhat simplified manner, by the following steps:

1. Select an example to be generalised. If no more examples exist, stop.

2. Construct the most specific clause, within provided language restrictions, which entails the selected example. This is called the ”bottom clause”.

3. Find, by searching for a subset of the literals in the bottom clause, more general clauses. Choose one with the best “score”.

4. Add the clause found in the previous step to the current theory, and remove all clauses that have been made redundant (it is worth noting that the best clause may make other induced clauses, not only basic examples, redundant). Move back to Step 1.

An important part of this thesis is our investigation of how to represent Deductor's knowledge base in a way accessible to the PROGOL algorithm. In particular, our goal is to analyse the relationship between the quality of learning and the amount of domain specific knowledge put into data transformation between Active Logic and PROGOL.

In Chapter 8 we describe our experiments which illustrate how different representations of Deductor's knowledge base influence learning results. In particular, we show that a small amount of additional domain specific knowledge needs to be provided in order for learning to be successful. One of the problems is the closed world semantics used by most ILP algorithms, including PROGOL. Deductor, in order to deal with incomplete knowledge that the agent has about the world, employs open-world semantics — from the mere fact that the agent is unable to prove something it does not follow that it is false.

For example, our agent reasons using predicate $Wumpus$. In particular, $Wumpus(b2)$ means that Wumpus is located on square $b2$. The agent starts the game with knowledge that:

$$Wumpus(a3) \vee Wumpus(b2) \vee Wumpus(b3) \quad \vee$$
$$\vee Wumpus(c1) \vee Wumpus(c2) \vee Wumpus(c3)$$

Intuitively, squares $a1$, $a2$ and $b1$ are excluded since the agent immediately observes that it does not smell on $a1$. At the same time, the agent initially knows neither “$Wumpus(b2)$” nor “$\neg Wumpus(b2)$” — it suspects Wumpus may be on this square, but it may also be somewhere else.

Under the closed world assumption employed by PROGOL, such a representation is impossible. Any time the agent does not know $Wumpus(b2)$, it is assumed that $\neg Wumpus(b2)$ holds. Therefore, for each such “uncertain” predicate, we introduce three alternatives in order describe all interesting possibilities. We do not, actually, *need* all three — theoretically, any pair

would be sufficient — but it would make learning much more difficult since good formulae would become more complex. In particular, we introduce three predicates describing the smelling phenomenon ($maybeSmells$, $noSmells$ and $knowsSmells$), as well as three predicates that describe the potential positions of Wumpus ($maybeWumpus$, $noWumpus$ and $knowsWumpus$).

6.6 Conditional Partial Plans

One question is how to represent situations and plans in the way most suitable for learning. We have decided to use ideas from Situation Calculus and describe every conditional branch of a plan separately, while we simultaneously allow quantification over them.

In both our experimental domains, the *first* step of a plan is an unconditional one — the agent simply decides how to move in a given situation. For Wumpus, the rest of the plan can have at most two branches (called *ifSmells* and *ifClear*, with *ifSmells* being taken if and only if it smells on the newly-visited square). For Chess, there are three explicit branches (each specifying the expected move of the opponent and the agent's response, without any meaning assigned to their order) and, additionally, a *default* branch, which will be executed whenever the opponent makes any move other than those three. It is our belief that such a representation is sufficiently general to be usable across many different domains.

Taking a very simple example from the Wumpus domain, the predicate $Position(\mathbb{P}, start, a2)$ means that in the starting position of plan $\mathbb{P}$, the agent occupies square *a2*. We utilise the predicate $Position$ to describe the details of the plan being discussed, in particular to describe what the possibilities of the agent moving around are. We introduce five different branch names, in order to provide the necessary flexibility of this description:

- $Position(\mathbb{P}, start, a1)$ — agent starts at $a1$.
- $Position(\mathbb{P}, visit, a1)$ — agent is guaranteed to visit $a1$, but it can happen at any time during execution of $\mathbb{P}$.
- $Position(\mathbb{P}, intermediate, a1)$ — agent is guaranteed to visit $a1$, but it is an intermediate position, it will not end up there.
- $Position(\mathbb{P}, ifClear, a1)$ — agent will end up on $a1$, but only if it *did not* smell on the first square it visited (i.e. if the *ifClear* plan branch gets executed).

- $Position(\mathbb{P}, ifSmells, a1)$ — agent will end up on $a1$, but only if it *did* smell on the first square it visited (i.e. if the *ifSmells* plan branch gets executed).

In addition, as mentioned earlier, we have introduced three predicates describing the smelling phenomenon ($maybeSmells$, $noSmells$ and $knowsSmells$), as well as three predicates which describe the potential positions of Wumpus ($maybeWumpus$, $noWumpus$ and $knowsWumpus$), all with board squares as arguments.

It is also important to note that it is not necessary to achieve 100% accuracy in our application. One interesting feature of our learning setting is that false negatives are not overly problematic: the point is to save some computations by discarding useless plans early, so if some bad plans are *not* detected, the worst that can happen is that some computations will still be wasted. False positives, on the other hand, are much more dangerous, since if Actor removes a useful plan from considerations, the overall quality of the solution can deteriorate. However, there is no way to express this distinction in PROGOL terms, so in this work we have decided not to separate accuracy into positive and negative parts.

6.7 Modes of Learning

When analysing the learning module, it is important to keep in mind that our agent has a dual aim, very akin to the exploration and exploitation dilemma, well-studied in Reinforcement Learning and related research areas. On one hand, it wants to win the current game episode, but at the same time it needs to learn as much generic knowledge as possible, in order to improve its performance at solving subsequent episodes.

Also, an important question is one of credit assignment, since the agent typically executes several partial plans before it reaches the terminal state of the game. Only the complete sequence of actions is then rewarded or punished. It can very well happen that one of the plans in a bad sequence was, in fact, good. There are numerous techniques being developed for dealing with this problem, each with its own advantages and disadvantages, and it is not clear at this point which one would best be suited for our particular case.

Finally, we have made some preliminary tests in a more *simulation-like* environment, where an agent executes a plan and observes its actual effects only. Therefore, it is prone to making mistakes when determining which plans

are potentially bad. Even though the learning algorithm we used allows for the possibility of noisy data, we have found that rather insufficient for our needs. Thus, the experiments we report here do not contain any noise — we demanded of Deductor to *prove* whether a plan is safe or not before it was used as a training example.

Finally, our agent faces an important tradeoff. Clearly, the longer the agent allows for the planning phase to proceed, the better plans it will get to choose from, and the more information about consequences of each plan will be known. On the other hand, more of the deduction effort will be wasted by considering potential situations which will not take place in this particular game episode, since they will be incompatible with actual observations.

Chapter 7

Module Interactions

In Chapter 2 we have introduced the architecture of our agent as a whole, mainly presenting a general overview of the system. We have also began to discuss the intended dependencies between various modules there, although it was by necessity done on a rather abstract level. Afterwards, we have devoted four chapters to describing, in detail, each of the modules constituting our architecture: Deductor, Planner, Actor and Learner.

In each of those chapters we have described the modules themselves, but we also kept analysing how each component interacts with other parts of the agent's architecture. What remains to be done here is to summarise and conclude those descriptions, clarifying once again the interdependencies among the modules, both in terms of information and control flow.

The main idea of our architecture is to ensure fruitful interactions between the modules introduced in previous chapters. Each of these modules roughly corresponds to one of the major subfields of Artificial Intelligence and is designed to accommodate the state of the art solutions that are developed there. Each module can provide a very good performance in specific situations and within appropriate limitations, especially if chosen with a particular application area in mind. Nevertheless, none of them is sufficient — alone — to achieve real intelligent behaviour of a generic situated agent.

The major contribution of this work is the idea of using multiple conditional partial plans as a way to exchange information (or knowledge) between modules. In this way, Planner comes up with plans that are potentially interesting, but it does not need to commit to a single one. Deductor reasons about each of those plans separately, but is also able to explore interactions or dependencies between them, as well as to make use of any similarities it can find in order to

extrapolate results concerning one plan to the others. Learner induces generic knowledge about what types of plans have been successful in the past, in what situations and under what conditions. Actor chooses, based on information provided by other modules, which plan should be executed in the future, as well as oversees the agent's operation and chooses the best course of action as soon as enough knowledge becomes available.

Using the architecture introduced in this work, our agent is able to reason about the *actual* state of the world, both about the details of the current situation and about the generic laws governing the application domain. Furthermore, it can also reason about the various *possible future* states of the world — namely, how it expects the world to change as a result of executing a particular plan.

Moreover, the agent reasons about the *plans* themselves, how successful they were in the past, both in general and in situations similar to the current one. Most importantly, it attempts to predict which plans are likely to be good right now, in order to focus its own limited computational resources on them.

Finally, the agent also reasons about its own knowledge and about its own available resources. This includes, among other things, evaluation of the knowledge needed for a particular plan to succeed, as well as the deadlines that are to be met by the system.

The Figure 7.1 illustrates in some detail the flow of information among the modules of the system. In particular, it makes explicit the knowledge base (occupying the central part of the picture) which stores all the beliefs the agent possesses at the moment. The contents of this knowledge base can roughly be divided into two major parts: beliefs about the world and beliefs related to the agent's internal state. The former part includes typical data often associated with situated agents, such as previous and current observations, generic knowledge, domain-specific definitions, description of current situation and hypothetical results of plan executions. The latter part is less ordinary, containing information like reasoning priorities influencing the deduction process, the historical data providing experience input to the learner, and rules governing plan selection by Actor.

Not unexpectedly, the most connected module is Actor, which maintains the control over the whole system. As mentioned earlier, this module also encompasses the reactive part of our architecture. In this sense one can see the correspondence between it and lower parts of a typical layered system. Both the *reactive part*, handling interaction with the external world, and a *conceptualisation part*, providing observation beliefs in symbolic format and maintaining their coupling to incoming percepts, are handled by Actor. This allowed us to

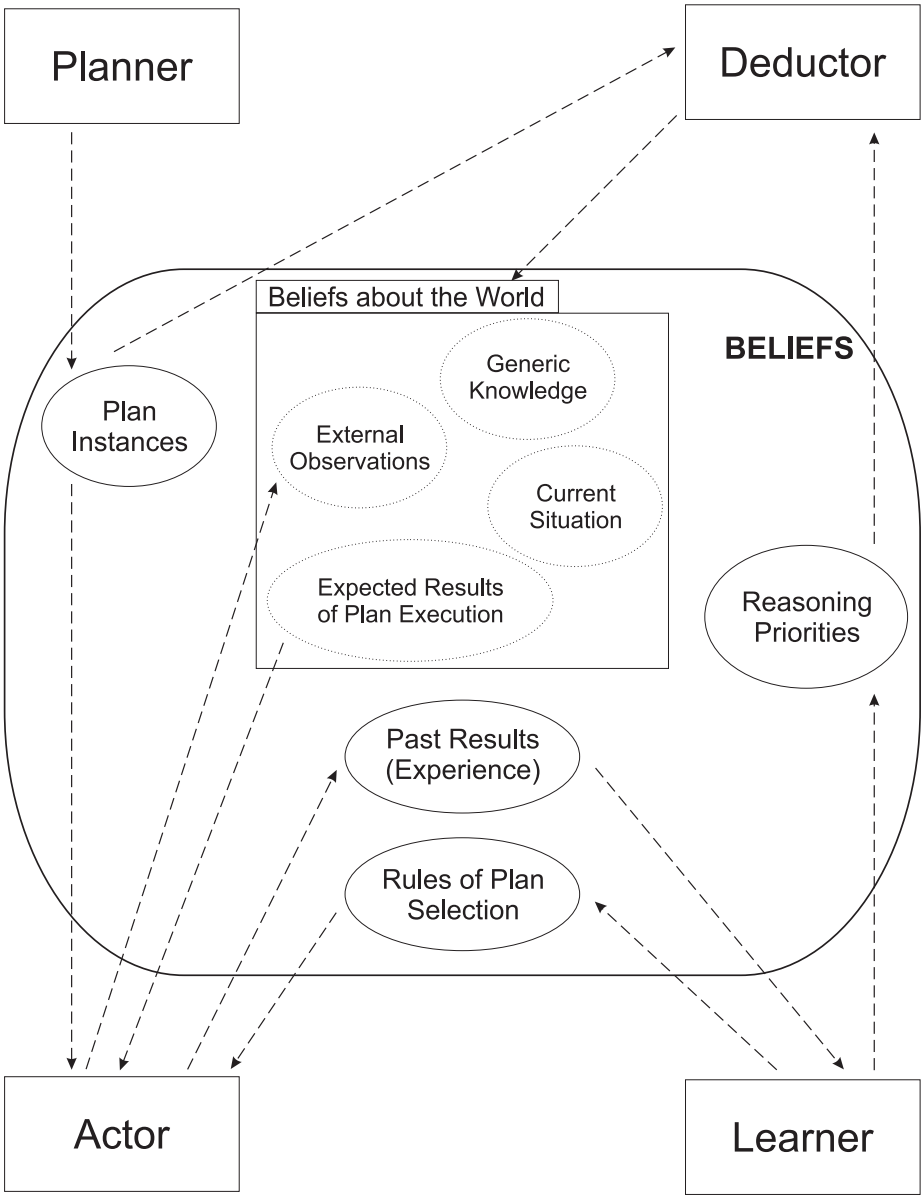

Figure 7.1: Information flow within the agent.

focus, in this thesis at least, on discussing higher levels of the architecture in more detail.

We have not devoted much discussion to the issue of our agent acting in the real world, concentrating on experimentations using a simulated environment instead. Our architecture allows both possibilities and it is only the reactive part that changes its *modus operandi*, from accessing the real world using sensors and actuators to accessing a simulation environment via some appropriate software interfaces.

Given our simple experimental domains this is not an issue, but in case of implementing the architecture on an autonomous robot it could be crucial. We fully expect to refine Actor in such case, possibly by defining some "mini-architecture" within this module, or by splitting it further. In particular, one can imagine that the robot could gather experience both by acting in the real world and in a simulated one *at the same time*. This way, provided that simulation holds sufficient level of accuracy, the learning process could be decoupled — to a degree — from physical actions.

In general, the major contribution of this work lies in the responsibilities of Learner and in the techniques used by it. Therefore, the relevant Chapters (namely 6 and 8) make up more than half of the entire thesis and the details regarding the functionality of other modules have been presented rather scarcely. It has been a deliberate choice made in order to avoid overwhelming the reader with details that would blur the actual focus of this work.

Although those details have been considered secondary for this presentation, they are of course very interesting from the point of view of actual implementation of rational, situated agent. Therefore we have provided, in Section 1.7, a reference to places where an interested reader can find this information.

Chapter 8

Experimental Results

8.1 Introduction

In this chapter we present experimental evaluation of several ideas we have introduced in this thesis. We have not yet implemented Deductor, Planner, Actor and Learner in full detail and with all the features needed for creating truly intelligent agents, since this is a task that will take a lifetime to accomplish. What we have, however, is enough to perform an evaluation of the general concepts we introduced in this work. In particular, we have a complete architecture and are therefore able to comment on performance and on some interesting interactions among various modules.

The major focus of these experiments have been on the usefulness of Machine Learning within the framework of our agent. Throughout this chapter we have used an ILP algorithm called PROGOL [Mug95] for all experiments (see Section 6.5 for details). This chapter is divided into four sections, corresponding to four different experiments we have performed.

In the first section (which is based on our paper [NM07c]) we present the results of learning to distinguish "dangerous" plans early. We show that PROGOL is able to find the correct hypothesis from as few as 30 randomly-chosen examples. Such a hypothesis allows the agent to save up to 70% of its reasoning time, since it does not need to waste resources analysing plans which turn out to be useless. Those results, however, require the user to provide additional domain knowledge specifically for the purpose of learning.

In the second section (which is based on our paper [NM07d]) we present a heuristic approach which allows the agent to extract such additional domain knowledge automatically. It turns out that the major problem with learning is

the *overwhelming* of PROGOL with all the knowledge possessed by the agent. Most of it is redundant, since it is a by-product of Deductor's reasoning procedure. A simple set of rules based on fuzzy sets allows the agent to filter out irrelevant knowledge, thus greatly improving the quality of learning.

In the third section (which is based on our paper [NM07b]) we present some modifications to the PROGOL learning algorithm which make it better fit the class of problems we are solving. This consists of three changes: the ability to directly take estimation of knowledge relevance as an input, to handle plan and branch parameters in formulae in a more efficient way, and to directly access agent's knowledge base expressed in Active Logic.

Finally, in the last section of this chapter, we combine those results together and show that our agent is able to learn throughout its lifetime and analyse how its performance improves as it gathers more experience.

8.2 Detecting Dangerous Plans

The first experiment we conducted concerns learning to detect "bad" plans early, as explained in Chapter 6. We have performed these initial experiments in order to evaluate the feasibility of our ideas and in order to check how well those ideas work in practice. Our focus was on interactions between modules and on showing that different approaches we combine do indeed complement each other.

In this section we describe our experiments by taking a special interest in how Deductor's knowledge base can be represented in different ways, in order to make it most accessible to the PROGOL algorithm (and, by extension, to other similar learning approaches). In particular, our goal was to analyse the relationship between the quality of learning and the amount of domain specific knowledge that the user needs to specify in order to have Deductor and Learner communicate effectively.

Obviously, the ultimate goal is to be able to solve this problem fully automatically, in a way which guarantees that whatever knowledge Deductor discovers, Learner is able to use it. As it turns out, it is not necessarily all that easy. If the user does not take learning into account at all, providing only the minimum of domain knowledge as required by the deductive part, the results obtained by PROGOL will be quite poor. In this section we investigate what is needed to make the learning process successful.

As a data set we have used three example runs of the Wumpus game on the 3x3 board. The agent had considered 134 plans in each of those episodes. This

is the total number of length-2 plans (both simple and conditional ones) in each case. Every episode consisted of four situations: the player started on square $a1$, first moved to square $a2$, then to square $b2$, and finally to square $b3$.

Each episode differed from the others by the position of the Wumpus. In the first run, the agent noticed that it smells on $b2$, and after moving to $b3$ and not dying, it figured out that Wumpus must be on $c2$. In the second episode, Wumpus was on square $a3$ and the agent discovered this fact after observing that is smells on $a2$ but not on $b2$. Finally, in the third episode, Wumpus was located on square $c1$.

For the Chess domain, we have used three board positions in which White have a winning strategy. We have hand-crafted 69 plans covering different types of situations, since it is obviously not feasible to analyse *all* possible plans in this domain (there are approximately 10^{15} of them). In the description below we will almost exclusively refer to the Wumpus domain, but the design of the Chess experiments followed the same principles.

In all experiments discussed in this section we have assumed that the agent has a perfect knowledge of which plans are bad and which ones are good. This means that it knows which *training examples* are positive and which are negative — there is no noise in the data. We have explained the rationale for this decision in Chapter 6.

In the first experiment, our goal was to use as little domain-specific knowledge as possible. In particular, we have not provided any *mode declarations* for PROGOL — the declarations for each background predicate were completely generic. The goal of the mode declarations is to reduce the hypothesis search space by limiting types of predicate arguments and by specifying which ones are input and which are output arguments and whether variables or constants should be used. There are heuristics that allow this information to be automatically extracted from the data, but they are not fool-proof and not always domain-independent. We have also decided not to filter the agent's knowledge in any way, except for removing Active Logic-specific extensions. In other words, the complete knowledge base of the Deductor has been used as an input to PROGOL.

Each training example corresponded to one of the plans the agent was considering, and each was labelled as being either good or bad. Each plan was described by the complete knowledge the agent had about it *at the moment the plan was suggested by Planner*. It is crucial to understand that this knowledge did not include the effect of deductive reasoning performed by Deductor — the point of learning is, in this case, to distinguish plans that are *not worthy* of being

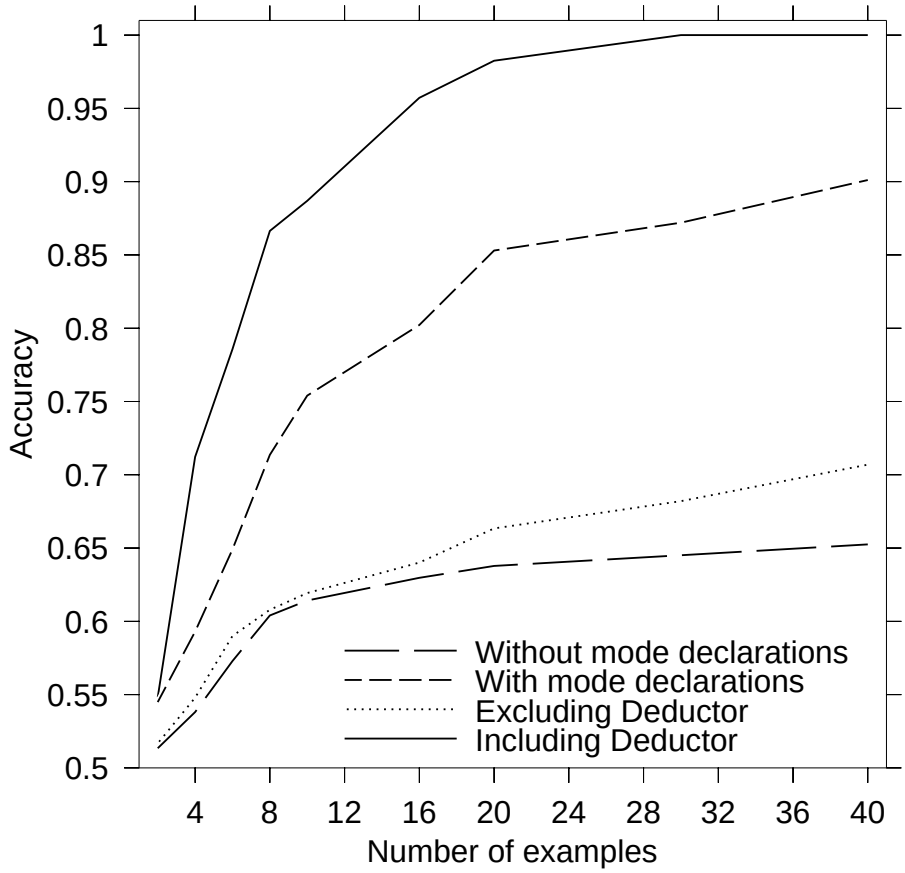

Figure 8.1: Results of learning in Wumpus domain.

reasoned about. Clearly, such classification is only useful if it is done *before* the reasoning is actually performed.

The following setting was used for this experiment, as well as for all the subsequent ones reported in this section. We have run PROGOL 500 times in total, varying the number of training examples used as an input. We started with one positive and one negative example only, and continued to increase their number until we reached 20 positive and 20 negative ones (using a different number of positives than negatives did not lead to any interesting results). For each size of the training set, we have made 50 runs, selecting examples at random, inductively generating a new hypothesis and calculating its accuracy.

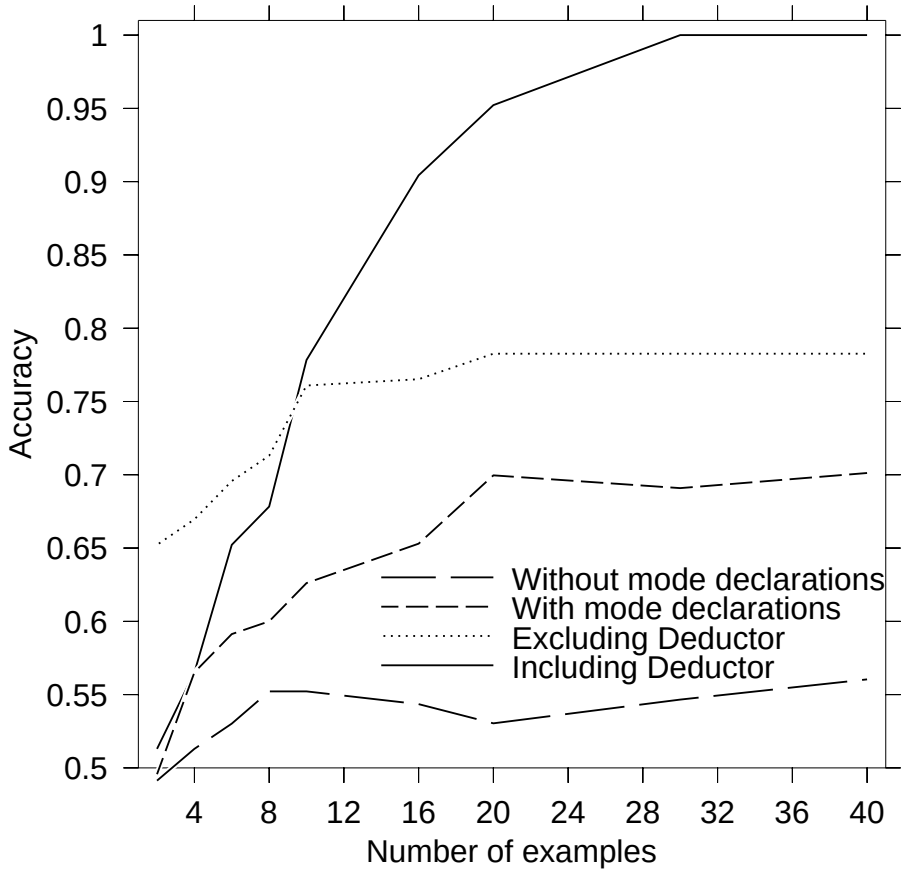

Figure 8.2: Results of learning in Chess domain.

We have plotted the accuracy[1] of the learned hypothesis as the lowest curve (marked "Without mode declarations") for Wumpus in Figure 8.1 and for Chess in Figure 8.2. It can be easily seen that the learning quality is too low to be useful in practice. This result is not really surprising as it is a well known fact that specifying appropriate mode declarations is very important for PROGOL. Without them, the heuristics that guide the search algorithm are next to useless. Nevertheless, this gives us a baseline with which we can compare further results.

[1]Defined in the usual sense, as the probability that a randomly chosen example (from the complete universe, not only from training set) will be classified correctly.

The second curve (marked "With mode declarations") clearly shows that providing even a very small amount of domain knowledge (mode declarations are very easy for a domain expert to specify) is enough to greatly improve the quality of learning. For example, for the three predicates describing the smelling phenomenon ($maybeSmells$, $noSmells$ and $knowsSmells$), as well as for the three predicates describing the potential positions where Wumpus can be ($maybeWumpus$, $noWumpus$ and $knowsWumpus$), we have specified the following mode definitions:

$$modeb(9, maybeSmells(+plan, -square))?$$

It means, basically, that the $maybeSmells$ predicate takes two arguments, the first being of type $plan$ and the second being of type $square$. It also means that the first argument is an input one (thus $+$) while the second one is an output argument (thus $-$). This predicate also has *recall* equal 9, which means that the learning procedure should explore at most nine different variable bindings.

Similarly, for the predicate $Position$, we have used the following:

$$modeb(18, Position(+plan, \#branch, -square))?$$

which means that the $Position$ predicate takes three arguments, and that the second one is of type $branch$ and it must be a constant, not a variable (thus $\#$). The first argument is a plan and the third is a square.

It can also easily be seen that the accuracy of the Wumpus domain is significantly higher than that of the Chess domain. Nevertheless, the learning is still not fully successful — even though all the knowledge theoretically needed for expressing the correct hypothesis is available. The most important reason for that is that there is too much data and the search space is too large for PROGOL to handle it sufficiently well.

Because of that, we have looked into ways of limiting the amount of knowledge used for learning — apparently, presenting all of the agent's knowledge to the ILP algorithm is not the best idea. We have decided to perform two more experiments. Within the background knowledge that we have identified as relevant for the concept of bad plans, there were two separate components: information about squares where it smells and information about squares on which Wumpus might hide. In principle each one of them, by itself, contains enough information to express the target concept.

Therefore, in the third experiment, we have decided to use only the initial domain definition and the observations that the agent made in previous situations. In particular, we have only used the predicates $maybeSmells$, $noSmells$

badPlan(A)	$\Leftarrow$	position(A,intermediate,B), maybeWumpus(A,B).
badPlan(A)	$\Leftarrow$	position(A,ifSmell,B), maybeWumpus(A,B).

Table 8.1: Correct definition of bad plans for Wumpus.

badPlan(A)	$\Leftarrow$	notProtected(A,default,rook), distanceTwo(A,default,black-king,rook).
badPlan(A)	$\Leftarrow$	isStep(B), position(A,B,rook,C), canMove(A,B,black-knight,C).

Table 8.2: Correct definition of bad plans for Chess.

and $knowsSmells$ for the Wumpus domain and mostly $Position$, $canMove$ and some geometrical relations for the Chess domain. The results of learning can be seen on the curve marked "Excluding Deductor" in Figures 8.1 and 8.2, so named since this setting roughly corresponds to an agent who does not have a specialised deduction module and attempts to learn from raw observations only. As it turns out, expressing the notion of bad plans using only those three predicates still proved to be too difficult for PROGOL.

As can be seen, the results in the Wumpus domain are pretty discouraging, while in the Chess domain the accuracy is actually *better* than when we have provided the full knowledge. This is caused by the fact that the Chess domain is much larger and much more complex, and removing almost *anything* from the knowledge base improves the quality of the hypothesis PROGOL is able to find.

In Wumpus, however, the learning algorithm is actually able to make some use of the extra knowledge provided by Deductor, while not quite being able to duplicate its work. This result reinforces our belief that the multi-module architecture we are developing is a useful one.

Therefore, for our fourth experiment, we have selected only the most relevant parts of knowledge generated by Deductor and presented them to PROGOL. In the Wumpus case this included both the smelling information ($maybeSmells$, $noSmells$ and $knowsSmells$) and the Wumpus position information ($maybeWumpus$, $noWumpus$ and $knowsWumpus$), while in Chess it included the $notProtected$, $distanceOne$, and $distanceTwo$ predicates. As illustrated by the curve marked "Including Deductor" in Figures 8.1 and 8.2, the agent managed to identify perfectly the bad plans from as few as 30 examples

Wumpus position	Full time (hours)	Improved time (hours)	Time decrease (percent)
c2	16.07 h	4.41 h	72.58
a3	14.72 h	5.52 h	62.49
c1	15.23 h	7.18 h	52.84

Table 8.3: Usefulness of learning.

chosen at random, in both domains.

The exact hypotheses that PROGOL learned are presented in Table 8.1 for Wumpus and in Table 8.2 for Chess.

It is interesting to note that as few as five *hand-chosen* example plans suffice for PROGOL to learn the correct definition for the Wumpus domain, which opens up interesting possibilities for an agent to *select* training examples in an intelligent way. This is an emerging area of research by itself, often called Meta-Learning (see for example [GVB04]).

Having established that successful learning is possible, one more thing that needs to be shown is whether it is actually *useful*. In our implementation (which is designed for flexibility of reasoning rather than speed) analysing a complete game of Wumpus (depending on the monster's real position) takes on the order of 15 hours. If Actor knows how to identify bad plans and forces Deductor to ignore them, the total time drops dramatically — to about *six hours*, as can be seen in Table 8.3. This is a clear confirmation of our claim that the knowledge gained due to learning from experience can be very useful in improving the efficiency of reasoning.

These results show that Deductor provides knowledge that significantly improves the quality of learning. At the same time, as mentioned above, learning the right hypothesis allows the agent to save a lot of its reasoning effort. Additional experiments reported in subsequent sections reveal even more synergy between modules in the architecture.

Finally, we would like to point out that the PROGOL algorithm, while a very efficient one, is not perfectly suited for the class of problems we are facing. It is sufficient for a proof of concept and to show the general usefulness of learning as such, but there is a number of reasons why an approach more specialised towards plan evaluation would be significantly more efficient.

8.3 Estimating Relevance of Knowledge

8.3.1 Introduction

The outcome of our experiments, reported in the previous section, was very promising. We have shown that learning can be both successful and useful for our agent. The major problem we have encountered before was the overabundance of irrelevant and redundant knowledge. This required the user to provide additional domain-specific information, which could filter formulae believed by the agent and only use the most relevant of them for learning.

In this section we investigate ways for the agent to automatically perform this task. The goal is to design mechanisms for limiting the amount of knowledge presented as an input to the learning algorithm and to do this in a way that is roughly based on the intuitive concept of "relevance of knowledge" — but is formalised in a way accessible to the agent.

An important obstacle is the lack of a satisfactorily objective measure of the relevance of knowledge. This field of study has seen a lot of interest in the past years, but the major problem has often been the lack of a formal model of what such *relevance* really means.

Typically the most interesting insights from the logical perspective are based on the *equivalence* relation, where one can say that two sets of formulae denote exactly the same models. This level of abstraction, however, is not quite appropriate for us, since it does not capture the qualities we are interested in: two theories equivalent in the semantic sense can differ greatly in how easy it is to generalise from them and to learn new concepts.

Similarly, from the philosophical point of view, there are equally few results we can use, since most of them are neither formal enough to be usable by computational agents, nor sufficiently self-contained. Relying on common sense knowledge is not going to work in our setting, since artificial agents are notoriously bad at employing it.

Our primary goal in this experiment is to explore how Deductor can automatically choose some parts of its knowledge in such a way as to maximise the quality of learning. Within the bounded computational resources that the agent possesses, it is not feasible to blindly generalise from the complete knowledge base of the agent. The generalisation process is inherently difficult, with a very high branching factor and therefore limiting input data is of crucial importance for realistic applications.

The major perspective in this section is on allowing the agent to learn efficiently. We have, however, noticed that there is a correspondence (whether

direct or indirect) between knowledge which is *good* for learning and knowledge which appears to be "naturally relevant" to human experts.

The exact limits of applicability of the solutions we propose here are still to be evaluated, but we do show that they are useful in our setting. At the same time, we are convinced that theoretical discussions presented here touch a number of important and difficult topics — we therefore believe they will be interesting for other researchers.

8.3.2 Intuitions

Estimating the relevance of a particular formula to the agent's task at hand is a very difficult problem and one that is, in general, unsolvable. In practice, however, there are certain regularities and conventions that human experts use when encoding domain knowledge for the agent, and it is our belief that some of those conventions can be exploited by the agent.

In a similar manner, there is a wealth of interesting information to be found in the deduction tree (trace) of the agent's knowledge base. By analysing the way in which the agent came to believe two particular formulae, a good guess can be made as to whether one of them is inherently more relevant than the other, whether one of them renders the other one obsolete, or whether they are more interesting together than separately.

The basic assumption is that human experts who create the agent's initial knowledge base, do it in such a way as to maximise its usefulness to the agent (*modulo* mistakes and ignorance, of course). Humans can often reasonably easily determine which parts of the knowledge base are most relevant for solving particular problems, and this is often reflected in the encoded knowledge in many ways. The ability to extract such hints would be very valuable to any rational agent.

It is important to notice at this point that we are not aiming for a precise, fool-proof and formal way of dividing an agent's knowledge into "relevant" and "irrelevant" portions. We merely need to reduce the input of the learning algorithm *sufficiently much* to ensure a successful operation.

We have considered several qualities which can be used as hints that a particular piece of knowledge is "good". We discuss and rationalise them here below on a rather abstract level, while in the next subsection we will show, in a more concrete form, an example of an implementation using those intuitions.

Derived from observations. For logic-based rational agents, there are typically few observations (because the cost of acquiring them and reasoning about

them is high), so it is unlikely that they are irrelevant. It is a wise idea, therefore, to try to keep them in the training knowledge whenever possible. Also, it is almost universally the case that the right decisions for an agent *do* depend on the acquired observations.

It is also important to notice that quite often it is not the raw observations themselves, but rather aggregations of them, together with the rest of the agent's knowledge, that form the most desirable learning input (for example, $WumpusPosition$ predicate should be used rather than $Smells$, and $CanMove$ rather then $Position$).

Consistency between plans. When looking at plans that are applicable in a given situation and analysing their expected outcomes, in many cases similar formulae occur. These sometimes encode exactly the same knowledge, and sometimes a different (even opposite) one. For example, it is possible that $Wumpus(a2)$ would be known from executing some branch of plan $\mathbb{P}_1$, and $\neg\, Wumpus(a2)$ would be known from executing some branch of plan $\mathbb{P}_2$.

For the logical sentences most relevant for the current problem, one can find multiple situations where the same knowledge holds across a wide spectrum of plans being considered. Having such consistency often enough is one hint that the given class of formulae is actually correlated with the agent's actions.

Of course, interesting formulae are not always like this, for many situations the values of any given predicate *need to* vary, which corresponds to the case where the real value is not known and the agent is actively exploring different possibilities. It is important that both ends of the spectrum are represented sufficiently.

Inheritance chains. We assume that the initial domain knowledge is chosen in such a way as to maximise the agent's performance, therefore one would expect that the *final* result of the reasoning process is likely to be the most important one. One way to capture this would be to assume that the more difficult a sentence is to infer (measured, e.g., as the number of steps needed to deduce it), the more relevant it is. This corresponds to the idea that such formulae "contain" more knowledge than easier ones.

This, however, would be overly susceptible to even small bits of irrelevant knowledge being present in the expert's description of the domain. On the other hand, in the reasoning traces of our agent, we have noticed a related and quite interesting pattern. It is often the case that the deduction tree of some formulae is rather degenerate: it contains a single branch which is very long, while all the others are very short. We have decided to call such a tree a *chain*. The middle sentences in such chains are often very boring indeed, with the end ones

containing everything of interest.

Similarities between plan branches. When looking at conditional plans, there is usually a number of branches that correspond to the different possible observations that can be made during execution. Looking for regularities between such branches can provide an interesting insight into which parts of the agent's knowledge base are the most relevant for choosing the best plan.

However, since this is too closely related both to how the agent treats observations and to the consistency between plans, we have not studied it further at this point.

Axioms are boring. Rather obviously, learning directly from the domain axioms is not the best idea. It is important to take advantage of the results of the reasoning the agent has performed, and often there are aggregations available that express the desired properties in ways much cleaner than barebone axioms alone.

One exception here, however, is a case when some axioms are never used in the deduction. If a human expert provides a number of formulae which are useless for reasoning, then the agent can reasonably expect them to play a role in learning.

Explore time differences. Given that our agent reasons using Active Logic, we have one more important clue regarding the interdependences between pieces of knowledge: the differences in step numbers of when particular formulae have been deduced.

One common pattern of reasoning is confined to a short time interval, with sentences inferred in the previous time step being used in the current one, and so on. On the other hand, an older formula is sometimes used in conjunction with the newest one and we believe that this pattern hints at something interesting. When an old piece of knowledge is relevant to the current situation, it is more likely that the same knowledge will be useful again in the future.

Another interesting idea would be to assume that things deduced earlier are more likely to be *generic* knowledge, therefore if they are applicable now, they will be applicable again. At the same time, a large percentage of the most recently inferred formulae will be specific ones, fitting the current situation but being useless anywhere else.

Smarter rules of inference. A big appeal of Active Logic is its powerful mechanism of adding domain-specific extensions to the well established sets of inference rules. Typical examples make use of timestamps and observation functions, but it is also possible to, among other things, specify that some rules of inference are more likely to lead to relevant sentences (for example, *modus*

ponens is interesting while *negation introduction* is boring).

Limited in number. On a more technical note, having too many formulae presented to an ILP algorithm, even if most of them are "good", means that it will fail to learn anything useful anyway (or, at the very least, that the generalisation process will be very fragile).

When the heuristic search employed by PROGOL is not able to meaningfully analyse the majority of the hypothesis space, the resulting knowledge will pretty much be a random set of clauses that fit the learning examples to some minimal degree. With the number of degrees of freedom that ILP algorithms have, such results are typically completely useless — and it depends greatly on many details of the training set, meaning that even very slight changes can result in completely different hypotheses.

Covering of the domain. The above qualities mostly analyse sentences in separation, but it is important to also look at the bigger picture. An important issue is that a large portion of both the initial and current knowledge base "contributes" to the learning process. The assumption throughout our work is that the domain knowledge is basically expected to be useful, so whichever knowledge we decide to present to PROGOL, it should have at least some chance of being influenced by all parts of it.

There are many possible definitions of what such an "influence" might entail, but the most obvious thing is to track which formulae were used to deduce the set we are interested in (akin to a very basic truth-maintenance system). Of course the fact that sentence β is deduced from α does not necessarily mean that the *rationale* for including α in the domain description has also been captured by β, but it at least indicates a possibility.

In general, it is often the case that many possible sets of sentences are in fact equivalent (in the classical logic sense), and therefore they "cover" the domain equally well. It is not necessarily obvious that the right way to choose among them is to analyse the actual reasoning process of the agent, but it is one of the approaches that seem intuitively viable.

Finally, an important decision is whether to consider each formula separately, or rather to cluster them in some way. We have decided to cluster sentences according to the predicates they contain. It can happen, for example, that the $Neighbourhood$ relation between two specific squares is interesting and important for learning, while the same relation between two others is irrelevant. There is a serious risk, however, of significantly distorting the domain knowledge, to the degree where the resulting hypothesis will generalise very badly. We have decided to forfeit some of this expressiveness and we only make

decisions as to which predicates are to be included in the training knowledge (i.e. if we decide that $Neighbourhood$ is relevant, then *all* formulae containing $Neighbourhood$ will be used for learning).

We feel the need to stress that our work here is of a heuristic nature (in the colloquial sense of the word) and that the rules we present can be easily fooled. It is not our aim, however, to present a bulletproof system, but rather to explore the natural clues left by human experts. We are aware that knowledge bases can be created so as to defeat our efforts, but we also believe that this does not happen naturally.

8.3.3 Implementation

In order to check those assumptions, we have implemented a system for evaluating subsets of knowledge according to those guidelines. To restate, it turned out that PROGOL was unable to perform a meaningful generalisation because it was given *too much* knowledge. Therefore, in order to obtain good learning results, we needed to specify which parts of the knowledge base are the most *relevant* ones.

The results of learning we have achieved can be seen in Figures 8.1 and 8.2, for *Wumpus* and *Chess* domains, respectively. It can be easily seen that selecting the right subset of the whole knowledge base can be very beneficial. However, in the experiment reported in Section 8.2 such selection had to be done by hand. Our goal right now is to automate this process.

We start with qualities that can be evaluated on a per-formulae basis, i.e., those where we can estimate — for each formula — to what degree it can be considered a "good" one. One of the most successful formalisms for dealing with this kind of problem, in particular one allowing for the systematic aggregation of several independent criteria, is the *fuzzy sets* approach (for introduction see for example [YZ92]).

We analyse each of the points from Section 8.3.2 in turn, presenting a formal way of calculating it.

Derived from observations. For every formula, we count the number of *observations* in its complete deduction tree and calculate the membership value according to the function presented in Figure 8.3 (1).

Consistency between plans. For every atomic formula (i.e. in the form $Predicate(args)$) we look at all the other plans in the same situation, and determine the ratio of those which contain $Predicate(args)$ to those which contain $\neg Predicate(args)$. We then calculate the membership value according to the

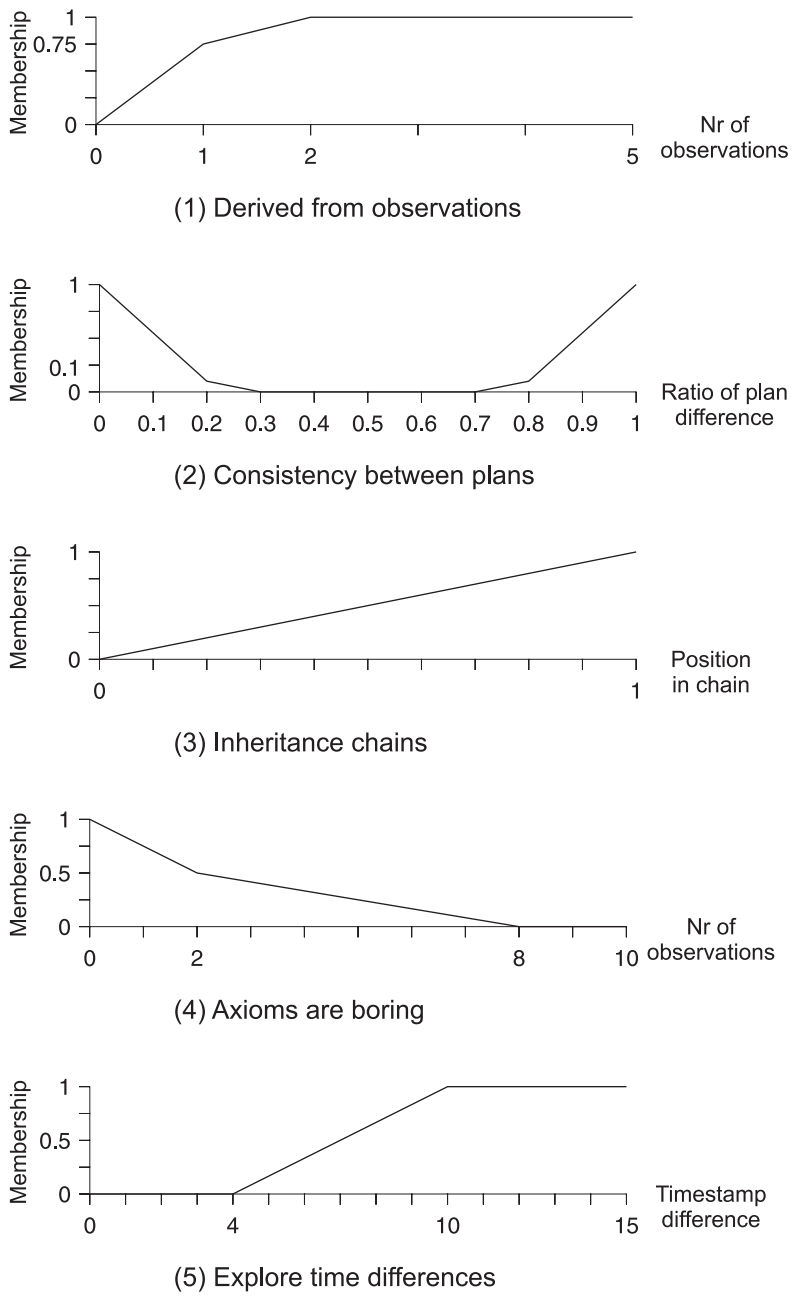

Figure 8.3: Fuzzy set membership functions.

function presented in Figure 8.3 (2).

Inheritance chains. For every formula we analyse its deduction tree and check whether it is a *chain* (i.e. a very unbalanced tree). If yes, we calculate the ratio of the formula's position in it to the total length of the chain. We calculate the membership value according to the function presented in Figure 8.3 (3).

Similarities between plan branches. Since we have not found a way to formalise this which would be simple and yet sufficiently distinct from other criteria, we have decided to skip it.

Axioms are boring. For every axiom, we count the number of other formulae which are inferred from it, both directly and indirectly. We calculate the membership value according to the function presented in Figure 8.3 (4).

Exploring time differences. For every formula, we check the timestamp of the oldest of the sentences it was inferred from, excluding initial domain axioms, and subtract it from the timestamp of the formula itself. We calculate the membership value according to the function presented in Figure 8.3 (5).

Smarter rules of inference. Since we have not found a way to formalise this which would be simple and yet sufficiently distinct from other criteria, we have decided to skip it.

To restate, we are interested in figuring out which predicates should be presented to the learning algorithm. To this end we evaluate all subsets of predicates and choose the best one (in Wumpus, there are only 6 predicates to consider, so analysing all subsets of them is not a problem). For any set of predicates $\mathcal{P}$, we start the evaluation by creating a set $\mathcal{F}_\mathcal{P}$ of all formulae in the knowledge base which contain only the predicates from this set.

We begin by calculating, for each subset $\mathcal{F}_\mathcal{P}$, the average value of membership function among all formulae in this subset, over all five of the above criteria. This tells us to what degree the current set of predicates fulfils each of the qualities we have identified. In order to aggregate all five of them into one number we have used the *product t-norm*. Even though the properties of this t-norm do not match the problem perfectly, using anything more complex is not justified given how approximate the parameters we use are.

The seventh quality, **limited in number**, did not really fit the fuzzy set approach. Besides, we have decided to use it as a normalising factor, since different sets of predicates can result in vastly different numbers of formulae being evaluated. Again, we have decided upon the simplest possible scheme: we divide the aggregate fuzzy membership value by the number of formulae. We call this value the *usefulness* of a predicate set.

At this point we are able to evaluate any predicate set, although this cannot

yet be used as the final measure, since the optimum value is always achieved by a set of cardinality exactly one[2]. This measure works quite well for separately estimating how good each predicate is on its own. It is very rarely the case, however, that a single predicate would be good for learning — there is almost always a need to have several different predicates.

The final step is to use the **covering of the domain** idea to arrive at a set of predicates that are *together* most likely to provide good learning results. Intuitively, we aim at adding predicates with high individual *usefulness* as long as they are reasonably independent of each other (i.e. as long as they originate from different "parts" of the agent's knowledge).

Therefore, we calculate the *coverage* of a set of formulae A as the ratio of all formulae in the knowledge base that appear in the deduction tree of at least one formula from A. We arrive at our final *relevance* score by multiplying the *usefulness* of a predicate set by its *coverage*.

We have implemented the above scoring method in a domain-independent way in our agent. We have run it on both the *Wumpus* domain and the *Chess* domain. As we hoped, the sets of predicates with the highest calculated *relevance*, in both domains, turned out to be exactly the ones we have earlier — by hand — identified as most relevant parts of the agent's knowledge. As reported in previous section, those sets of predicates lead to good learning results (see Figures 8.1 and 8.2).

It is not obvious how general our results are, since two example domains is too few to come to a definitive conclusion. It is enough, however, to suggest that our ideas have merit.

The exact scoring methods, as well as criteria used, are only an initial idea, and more work is needed. Nevertheless, the results appear to be pretty stable and small modifications to the above parameters do not influence the final result in any significant way.

8.4 Adapting Learning Algorithm

In this section we investigate some ways to adapt the PROGOL algorithm to the specific needs of the Learner module within our architecture.

PROGOL is based on the idea of *inverse entailment* and it employs a covering approach similar to the one used by FOIL [Mit97] in order to generate hypothesis consisting of a set of clauses that cover all positive examples and

[2]For any two predicates p_1 and p_2, the *usefulness* of the set $\{p_1, p_2\}$ always lies somewhere in between the usefulness for p_1 and for p_2.

do not cover any negative ones. It starts with a positive example e and knowledge base B and creates a set $\bot$ which is the set of all ground literals true in all models of $B \wedge \bar{e}$. Due to the properties of inverse entailment, the complete set of hypotheses $\mathcal{H}$ consists of all the clauses which Θ-subsume *sub-saturants* of $\bot$. PROGOL uses mode declarations to limit the size of $\bot$ and refinement operator ρ to efficiently search only a subset of $\mathcal{H}$. More details can be found in [Mug95].

We have made two modifications to the PROGOL algorithm in order to make it better suited for use in rational agents. Both of them were inspired by the difficulties PROGOL encountered when solving our problem, difficulties that apparently were the result of mismatches between its assumptions and the properties of the problem at hand.

We have also added the ability to directly access agent's knowledge, expressed in Active Logic, without an additional step of data transformation. We do not focus on this here, since it does not affect the results of learning in any way. It does, however, help to integrate learning closer with the rest of the agent architecture. Such direct access includes the transformation from *open world* semantics used by the Deductor to *closed world* semantics used by PROGOL.

8.4.1 Branch Awareness

First of all, it is well known that PROGOL is sensitive to the arity k of predicates in its background knowledge, since the cardinality of a sub-saturant set is bounded by n^k. Often this is not an issue, since the arity of predicates is typically kept low. In our setting, however, many predicates contain two additional arguments, namely the plan and its branch. Instead of simply saying $Wumpus(a1)$ to state the monster's position, we need to say $Wumpus(\mathbb{P}_1, \mathbb{B}_1, a1)$, i.e. in branch $\mathbb{B}_1$ of plan $\mathbb{P}_1$, Wumpus is on $a1$.

This has lead to a problem where PROGOL, in the Chess domain, was unable to learn the correct hypothesis in the form we originally wanted. We have initially specified a predicate $chooseBranch$ with mode declaration:

$$modeb(4, chooseBranch(\#step, -step))?$$

which is useful for "naming" a particular branch, i.e. binding it to a variable which subsequent predicates can later use. Moreover, we declared other predicates with the following:

$$modeb(8, notProtected(+plan, +branch, \#piece)$$

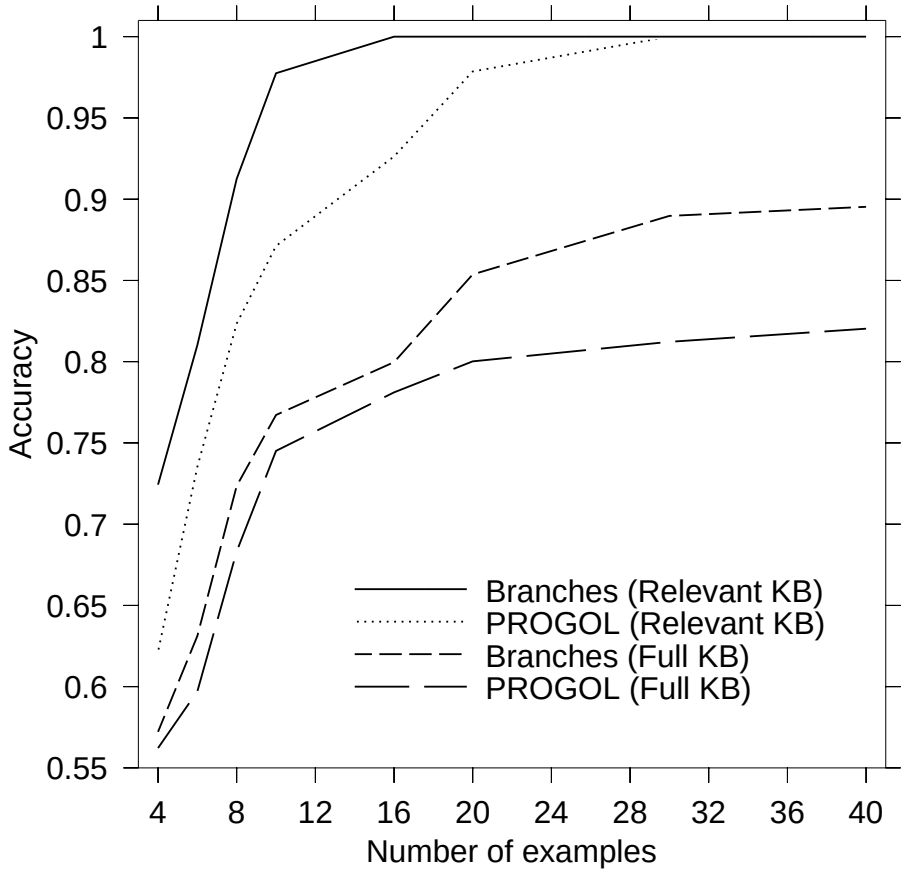

Figure 8.4: Branch awareness in Wumpus Domain.

This way we can guarantee that *the same* branch is used throughout the whole clause. In particular, we intended one of the clauses in the final hypothesis to be:

$$badPlan(A) :- chooseBranch(default, B), notProtected(A, B, rook), \\ distanceTwo(A, B, rook, black\text{–}king).$$

meaning "whenever in the default branch the opposing king is close to our rook, the plan is dangerous."[3] Such a hypothesis, however, proved to be too difficult

[3]Observe that in the default branch we do not know the exact move the opponent has made,

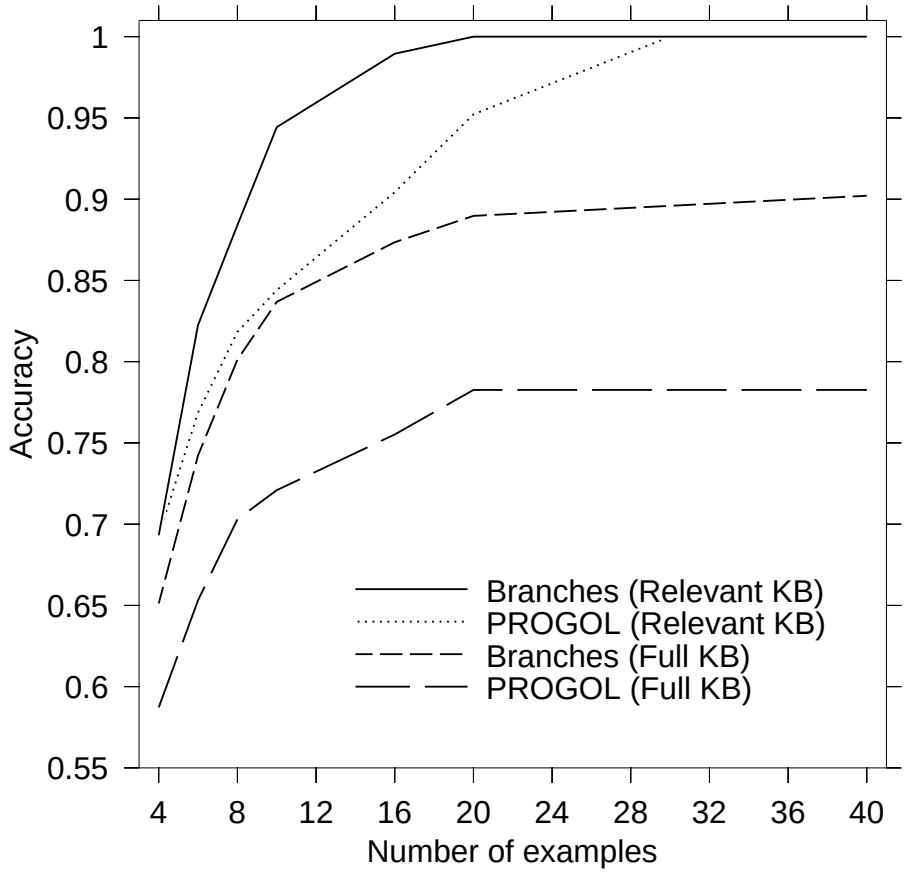

Figure 8.5: Branch awareness in Chess Domain.

for PROGOL and despite numerous attempts, we have not managed to "convince" it to learn it. We had to give up on the $chooseBranch$ predicate and settle for a similar, yet somewhat less natural, mode declaration:

$$modeb(8, notProtected(+plan, \#branch, \#piece)$$

This resulted in the hypothesis:

therefore $Position(black\text{–}king)$ is the "old" position and we need to be extra careful about our rook's safety.

$$badPlan(A) :- notProtected(A, default, rook),$$
$$distanceTwo(A, default, rook, black\text{–}king).$$

Notice that this one uses separate, unrelated constants as arguments of $notProtected$ and $distanceTwo$ predicates. In our experiments PROGOL required some additional examples before it learned not to use two *different* branches there. For example, one of the clauses it induced when given only eight training examples was:

$$badPlan(A) :- notProtected(A, b_1, rook),$$
$$distanceTwo(A, b_2, rook, black\text{–}king).$$

It is, obviously, not good — whether the rook is protected in some branch b_1 has no meaningful relation to the distance between rook and black king in b_2.

To fix this issue, we have modified the PROGOL learning algorithm to *transparently* hide branches, i.e. to automatically restrict knowledge base to only contain facts from a single branch when reasoning. This has lowered the arity of most predicates by one and allowed for better hypothesis to be found from fewer examples.

As a special case we have also introduced a branch called ANY, which combines the knowledge from *all* the branches of a given plan. This is useful for expressing knowledge of the kind "if there exists any branch b such that $\alpha(b)$, then ...".

It is questionable, however, whether this is not too drastic a measure, since one can imagine domains where different plan branches are not completely independent and it would be beneficial to reference two or more of them from a single hypothesis clause. If such a need arises, the correct way to handle it would be to enrich the mode declaration language. The above solution, however, is good enough for our current needs.

Figures 8.4 and 8.5 compare the results of learning using vanilla PROGOL against one which contains built-in knowledge about plans and branches. It can be easily seen that our modification leads to better learning results.

It is also important to note that the hypothesis learned by the modified algorithm is of better quality, since — as explained above — it contains a variable bound to the default branch, instead of having duplicate constants.

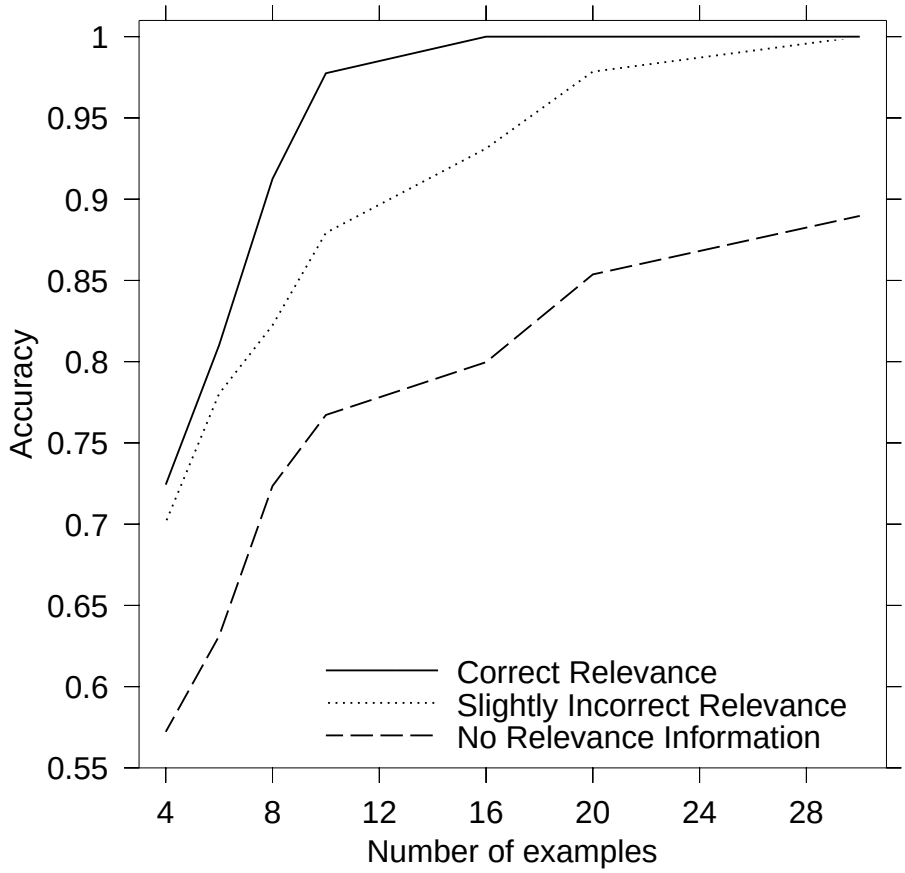

Figure 8.6: Correctness of Relevance.

8.4.2 Knowledge Relevance

As can be seen in the results mentioned in Section 8.2, too much knowledge can significantly decrease the performance of the learning algorithm. In a typical ILP setting it is not crucial to quantify which parts of knowledge are more and which are less relevant[4] — since an expert provides the complete knowledge in the first place, if she knows that some parts are irrelevant, she will simply omit

[4]Although in the general field of machine learning there is a lot of interesting work being done on this topic, especially with regard to data mining. Good surveys are, for example, [BL97] and [VD02].

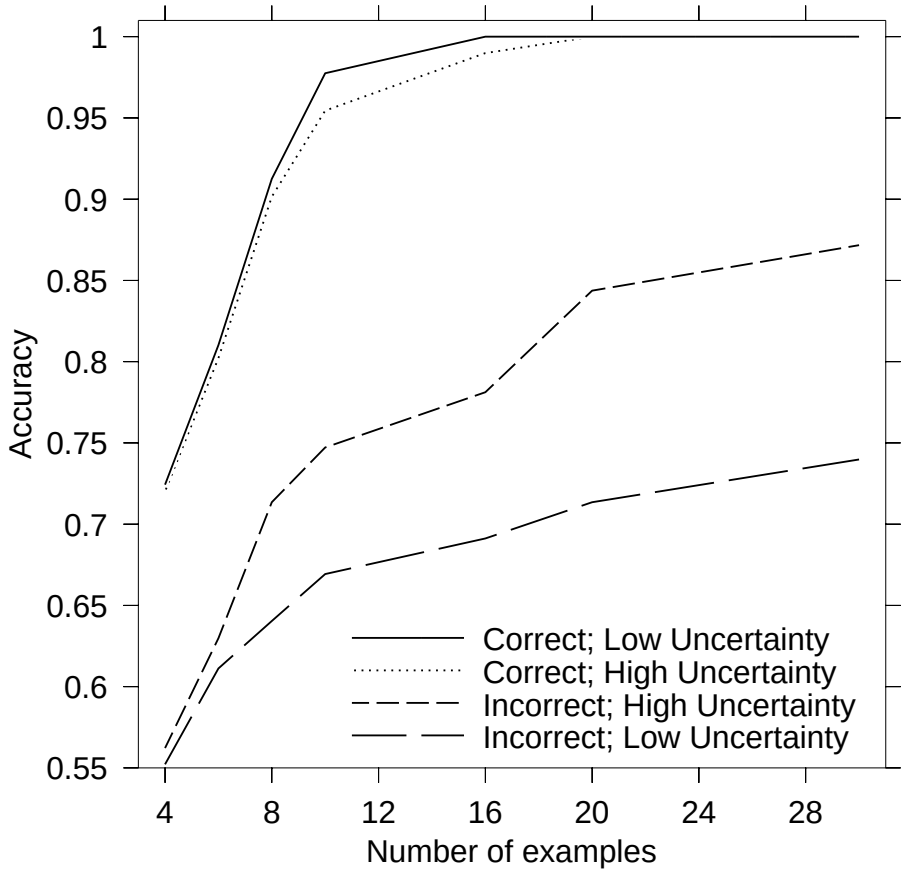

Figure 8.7: Uncertainty of Relevance.

them.

The situation is somewhat different in the agent setting, where the knowledge from which we are learning does not come *directly* from an expert, but rather from Deductor. It is a product of an initial domain description, the observations the agent has made, and its own deductive process — and all those elements interact in complex ways.

The amount of irrelevant knowledge in such a conglomerate is, typically, rather high. In Section 8.2 we have shown that choosing only the relevant parts can lead to much better learning results. In the previous section we have analysed how an automatic heuristic procedure can be constructed to select which

knowledge is most useful.

It is important, however, to stress that any kind of automatic procedure to estimate the relevance of data is very approximate in nature and even though it works very well on our domains, we do not want to *over-commit* to its results. In particular, the idea of completely removing the predicates that were deemed irrelevant from the *whole* learning process is very dangerous. The price of making a mistake seems to be too high in this case.

What we develop, instead, is a way for the learning algorithm itself to take into account the relative relevance of knowledge when it is building the hypothesis. In the case of PROGOL, the perfect place to take knowledge relevance into account is in the ρ operator, where the current candidate hypothesis is being generalised by new literals.

In Figures 8.6 and 8.7 we report our analysis of usefulness of building relevance considerations into the algorithm itself, in the Wumpus domain only.

On Figure 8.6, the curve "Correct Relevance" depicts unambiguously correct information about relevant predicates and corresponds exactly to "Branches (Relevant KB)" from Figure 8.4; the curve "No Relevance Information" depicts complete lack of relevance information and corresponds to "Branches (Full KB)". The most interesting one, curve marked "Slightly Incorrect Relevance", illustrates that even providing relevance information which is not entirely correct can be beneficial.

Figure 8.7 presents how the uncertainty of relevance information influences the results. Low uncertainty is the case when relevant predicates have the estimate of 1 and irrelevant ones have the estimate of 0. High uncertainty is the case where relevant predicates are estimated slightly above 0.5 and irrelevant ones slightly below 0.5. An interesting fact is that increasing uncertainty of correct relevance information lowers quality of learning, but not significantly. On the other hand, as can be expected, providing incorrect information with high certainty significantly disrupts the learning process.

8.5 The Agent Life Cycle

In this section we use the results of earlier experiments in order to build a complete situated, rational agent and to show how it can use learning in order to continue to improve its performance throughout its whole lifetime. Here we only use the Wumpus domain, since it is much more illustrative to the ideas we want to focus on.

Our main goal in this experiment is to take a look at the agent as a whole,

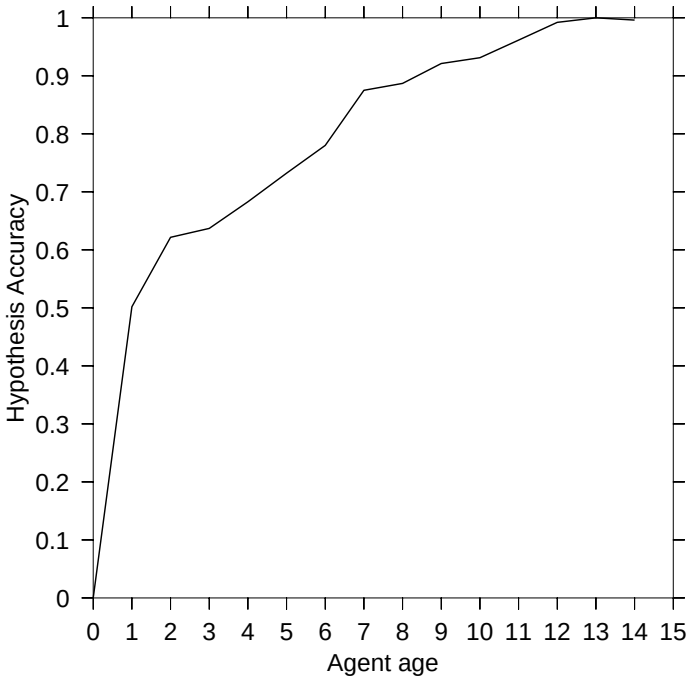

Figure 8.8: Average “badPlan” hypothesis accuracy.

and to analyse how it operates when solving problems continuously and how its performance improves as it gathers additional experience. As always, our agent starts with some domain knowledge and no actual “memory”.

Because of that, when Planner initially generates a set of plans, it has no way of knowing which of them are good and which are bad. Deductor needs to devote an equal share of time to each of those plans. This means that the agent is reacting slowly, since it wastes a lot of time on some plans only to discover that they may lead to losing the game and need to be discarded.

After each game, however, the agent gathers some experience and utilises Learner to create a hypothesis for predicting which plans can be discarded early. This means Deductor will now waste less resources and the overall response time of our agent will be much shorter. In order to illustrate our ideas better, we allow Actor to store only two plans per episode in its long-term memory. One of them is a (safe) plan from among those it executed, and the other is a (dangerous) plan it reasoned about and was forced to discard.

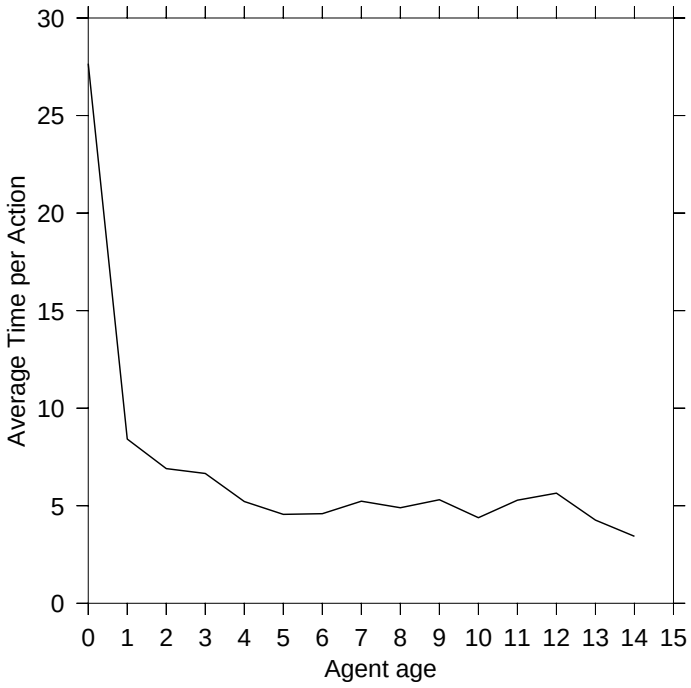

Figure 8.9: Average time per action.

In this setting, Actor actually chooses plans for execution randomly, from among all those considered to be safe. The main focus of this experiment is to evaluate actual usefulness of the learning algorithm that has been partially tested (in separation from other modules) in previous sections. Right now we analyse the architecture as a whole, showing that learning is, indeed, helpful to the agent.

The framework established here can be also used later to test various modifications and upgrades of the modules within the architecture. We are able to analyse whether it is more beneficial, for example, to use an expensive planning algorithm together with a dumb plan selection procedure, or rather to use a simplified planner and spend more time on plan analysis, evaluation and comparison.

In Figure 8.8 we show the hypothesis accuracy achieved by the agent as a function of its “age”, i.e. the number of episodes it has lived through. The age is directly related to the number of training examples Learner has at its dis-

posal, and therefore this figure corresponds very closely to the results reported in Section 8.2. Observe, however, that in the current experiment the training examples are no longer uniformly chosen from the whole set of possibilities — though this difference does not seem to influence the outcome in any noticeable way.

A more interesting result is presented in Figure 8.9. This graph depicts the average time (in seconds) it takes for the agent to analyse its situation and decide what to do. In other words, it represents how long the agent "thinks" before it concludes it is ready to act. As can be seen, it takes much longer for a "young", inexperienced agent to decide what to do, than it does for an "older", smarter one — because the latter has less plans that it actually needs to analyse.

It is interesting to see that while the reasoning time drops down dramatically at the very beginning, it remains relatively constant after a while. It might be tempting to say that there is no point in learning after step 4 or 5, but assuming that would be wrong. Figure 8.9 should be analysed together with the previous one, i.e. in the context of the accuracy of the learned hypothesis. While further learning does not speed up deliberation significantly, it definitely refines the hypothesis and makes it more accurate.

In other words, early hypotheses make the agent ignore about as *many* plans as the late ones, but definitely not *the same* ones. In fact, a number of plans they discard are actually good, and some that are kept are actually bad. This definitely influences how effectively the agent behaves in the world.

Finally, we have performed an experiment where the agent attempts to learn which plans are better than others. We model it by using a predicate $betterThan(\mathbb{P}_1, \mathbb{P}_2)$. In order to generate training data for PROGOL we have used a Monte Carlo simulation technique — the agent uses a *simulator* of Wumpus environment to stochastically evaluate quality of plans.

Given plans $\mathbb{P}_1$ and $\mathbb{P}_2$ and situation $\mathbb{S}$, the goal is to determine whether one of those plans is better than the other — observe that plans may very well be equally good or even incomparable. To this end, our agent repeatedly (in our experiments, 2000 times) executes $\mathbb{P}_1$ followed by a *random* sequence of (safe) plans, until it wins the game episode. The agent then calculates the average number of steps $\mathcal{T}(\mathbb{S}, \mathbb{P}_1)$ it takes to win the game from situation $Result(\mathbb{S}, \mathbb{P}_1)$. After calculating $\mathcal{T}(\mathbb{S}, \mathbb{P}_2)$ in a similar manner, the agent has a measure of plan quality: if $\mathcal{T}(\mathbb{S}, \mathbb{P}_1)$ is significantly smaller than $\mathcal{T}(\mathbb{S}, \mathbb{P}_2)$, then $\mathbb{P}_1$ is — most likely — better than $\mathbb{P}_2$.

Such technique, however, has a number of drawbacks, besides the obvious fact that it is computationally expensive. First of all, it is far from being reli-

able, since in many good situations there is only a small number of actions that quickly lead to the goal, while the majority of possibilities are misleading. By performing random actions, simulator is likely to never encounter the optimal path. Also, the measure itself is quite sparse, since if the values of $\mathbb{T}(\mathbb{S}, \mathbb{P})$ are close, they tell *nothing* about true quality of plans. Nevertheless, this technique turned out to be sufficient for our need, since we do not intend to use it for controlling the agent, but only to generate input data for the learning process.

It is important to point out that this experiment is for illustratory purposes only. A major problem is that the knowledge representation we use is not sufficiently rich to express the correct type of differences to determine which of two plans is better. Intuitively, it would require notions such as "better discriminate Wumpus's position" or "more knowledge is gained". However, neither Wumpus domain knowledge, nor hypothesis language used by PROGOL, contain the arithmetical axioms necessary to compare cardinalities. In fact, the best hypothesis our agent learned contained a number of clauses of the form:

$$\begin{aligned} betterThan(\mathbb{P}_1, \mathbb{P}_2) :- \quad & maybeWumpus(\mathbb{P}_2, noSmells, a3), \\ & notWumpus(\mathbb{P}_1, noSmells, a3). \end{aligned}$$

where $notWumpus(\mathbb{P}_1, noSmells, a3)$ means that if the "noSmells" branch of plan $\mathbb{P}_1$ is executed, the agent will know that Wumpus is not on $a3$.

Obviously, such clause is far from perfect, but it turns out that it (mostly) suffices to distinguish between plans which do provide *some* new knowledge and plans which provide *no* such knowledge.

In other words, given two useful plans, the agent is still not able to determine which one of them is better. However, given a useful and useless one (i.e. a plan that visits *some* new square(s) and gathers *some* new knowledge versus a plan which only traverses the already-explored part of the board), Actor is able to select the former one. And as shown in Figure 8.10, this is enough to significantly improve agent's performance.

Given a set of plans and some $betterThan$ relations that hold among them, our agent is finally equipped to select some plan and claim it as "the best one". Typically, $betterThan$ relations we obtain do not form a well-defined order, since cycles and disconnected components are common. Therefore, the agent simply rates each plan according to how many other plans it is better than: the one "dominating" the highest number is chosen for execution.

In particular, in this experimental evaluation, the agent employs the following algorithm:

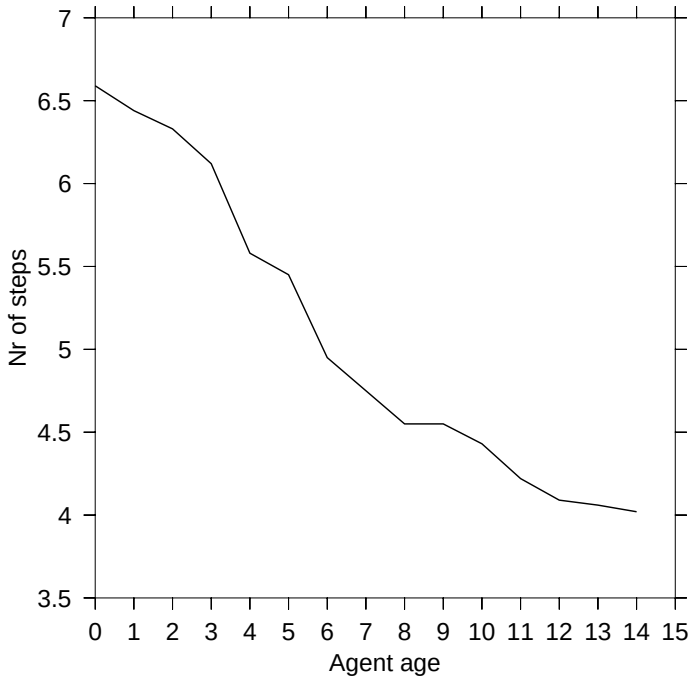

Figure 8.10: Average number of actions per game episode.

1. Generate all applicable plans — Planner.
2. Determine which plans are necessarily “bad” and discard them, based on Learner’s experience-generated knowledge — Actor.
3. Analyse all the remaining plans and determine which ones are safe — Deductor.
4. Choose “the best” among all safe plans, according to measure described above — Deductor.
5. Execute it, observing new smelling facts — Actor.
6. Analyse the new situation, determining if Wumpus’s position is already known — Deductor.
7. If not, go to point 1.

8. Choose one of the plans executed in this game episode, and one of the plans determined to be unsafe — Actor.
9. Add them to the agent's experience — Learner.
10. Run PROGOL, trying to improve the $badPlan$ hypothesis — Learner.
11. Compare all safe plans using the Monte Carlo simulator technique — Actor.
12. Add the best and the worst plan to the agent's experience — Learner.
13. Run PROGOL, trying to improve $betterThan$ hypothesis — Learner.

The results clearly indicate that our architecture works: the knowledge acquired via learning serves its purpose since it makes the agent behave more efficiently.

Exploiting previous experience is one of the most promising ways of coping with uncertainty and lack of knowledge about the agent's current situation. One of our main goals, and of the biggest challenges we face, is to ensure smooth cooperation between deduction and induction within the agent's framework. The results presented in this section show that this goal has been, at least partially, achieved.

Chapter 9

Conclusions

9.1 Thesis Summary

In this thesis we have presented a framework for developing situated, rational, resource-aware agents that are able to improve their performance from experience. Our architecture allows the agent to combine planning, deductive reasoning, inductive learning and time-awareness in order to successfully operate in dynamic environments. Agents create conditional partial plans, reason about their consequences (using an extension of Active Logic with Situation Calculus features), execute them and employ Inductive Logic Programming to generalise past experiences.

We have also reported on our experiments using the PROGOL learning algorithm to identify bad plans early in order to save the agent the (pointless) effort of deliberating about them. We have analysed how the quality of learning depends on the amount of additional, domain-specific knowledge provided by the user. Moreover, we have shown that it is possible to adapt a general purpose learning tool such as PROGOL to better fit the specific requirements of situated agents. Finally, we have presented that the Inductive Logic Programming approach works well within the architecture we are developing. In particular, we have demonstrated that successful learning can result in a dramatic decrease of the agent's reasoning time and in a significant increase of its performance.

In order to summarise our work, we would like to point out that the results described in this thesis come in two flavours. One is the conceptual work on developing an architecture for situated rational agents, with the major focus on using conditional partial plans. Many ideas are related to how responsibilities should be divided between the modules and how Planner, Deductor, Actor and

Learner should interact and cooperate in order to achieve agent's goals. Necessarily, this work has been discussed without going into too much detail on *how* this should actually be realised, partially in order to keep the focus of the thesis clear and partially because many of those issues still remain to be solved, being in themselves interesting research areas.

The other kind of results concerns the current implementation of the architecture. It has been tested on a toy domain and it employs a number of simplifications with regard to the ultimate concept of the system. Nevertheless, this implementation can be used to test and validate the architecture. Most importantly, though, the results of experiments performed using this implementation do — even if it requires some degree of optimism — suggest that our approach is viable. Obviously, it will still take a lot of work before our ideas become practically useful and our agent can outperform state of the art solutions in complex, practically useful domains. Nevertheless, we have shown that this line of research should lead to further interesting results.

In particular, we have high hopes with regards to the kinds of problems that our framework will be able to handle. It is our belief that a truly intelligent agent can someday emerge from it. Our architecture has — at least on the conceptual level — all the means necessary to deal with the most challenging problems situated agents face: limited resources, imperfect domain knowledge, incomplete and noisy observations. Rational, autonomous agents are the ultimate dream of Artificial Intelligence research since its beginning. We believe that our work is a small step towards achieving this dream.

9.2 Future Work

The research presented here can be continued in many different directions. The most obvious one is to improve the learning algorithm by making it aware of the actual meaning and origins of its input data. By exploring the difference between *fluents* and *non-fluents*, for example, a hypothesis can be found that will match different situations better.

The most important thing, however, is to focus our attention on how to handle the unavoidable uncertainty within training data. Again, there is a lot of work being done in this area, but there are some rather unique issues in the case of rational agents and we feel that it is important to design a solution that will be able to exploit whatever advantages can be found. The current setup assumes complete domain knowledge, while in many situations this assumption might be violated — for example, the agent might not know that, actually, the Wumpus

can move.

The second step is to devise an algorithm to efficiently "learn to compare", instead of creating a classifier. After all, the real goal of the agent is to choose *the best* plan available. Discarding bad plans is a step in this direction, but the classification approach is not necessarily the right one when the "least bad among dangerous" or the "most rewarding among marvelous" is to be selected. Our surrogate approach based on using $betterThan$ predicate has many drawbacks and, if possible, a better solution should be developed.

Moreover, the exact way of representing plans and their properties, for the sake of efficient learning, requires more work. Our current setup, which simply uses slightly modified Situation Calculus mechanisms, is most likely suboptimal — but anything better suited would need to have more support built into the learning algorithm itself.

One very promising idea seems to be the exploration of the epistemic quality of actions. An agent should pursue those plans which provide it with the most important knowledge. This would require assessing importance for a particular domain by observing effects specific pieces of knowledge have on agent's overall performance.

Moreover, in the case of rational agents we have the very interesting notion of "experiment generation", since they often do not learn from a predefined set of training examples, but rather face the complex *exploration vs exploitation* dilemma. How an agent should act in a way which both provides short term rewards (or, at the very least, keeps it safe) and at the same time offers a chance to learn something new is often far from obvious.

As mentioned earlier, we are also looking into discovering subgoals and subplans. It seems that one of the most useful capacities of humans is their ability to divide a complex problem and to solve each subproblem separately before combining their solutions into a global one. We would like to force our agent to discover this possibility.

Another direction we would like to investigate in the future is making the learning module discover general rules which extend domain knowledge. The ability to invent domain-specific reasoning heuristics would be very useful for rational agents. Finally, the current setup assumes a complete domain knowledge, while in many situations this may not be the case at all. The system should, if possible, allow the agent to learn domain knowledge so that it can complete its understanding of the environment. An example of such a rule might be, in the RoboCup domain, "do not perform a *kick* action unless you know that the ball is in front of you". It seems that availability of such rules

can save a substantial amount of work for Deductor, if it inductively discovers "reasoning shortcuts".

Another clear advantage would be to reuse a valid plan in a different context. As long as the context does not differ substantially, this operation should lead to a fast solution of a problem similar to one solved in the past.

Last but not least, we would like to investigate ways to incorporate interactions with a user. Domain experts can be an invaluable source of knowledge and the agent should be able to exploit this. For example, to better adjust tradeoff between spending time on deduction and induction, the agent could be guided by an external observer (the user) providing a feedback about its performance.

Besides the possibilities listed above, which mostly focus on Learner and its interactions with the rest of the agent, there is a number of open issues regarding deductive reasoning, planning and acting. An obvious step forward would be to replace our simple versions of Planner and Actor with state of the art systems, in order to ensure that the interfaces between modules we designed will work with existing software. It would also be interesting to find out what performance improvements can be achieved this way. Another work worth doing is to apply our concepts in different environment dynamics. An experiment of using our architecture to control a physical artifact (such as a mobile robot) would be the next step in showing advantages of this framework.

The list above does not cover all the possible further investigations and extensions of the proposed system. It is rather a reflection of the author's own interests which focus on the border between Machine Learning and other subfields of Artificial Intelligence.

Bibliography

[AG02] Robert Andrews and Shlomo Geva. Rule extraction from local cluster neural nets. *Neurocomputing, 47*, 2002.

[Ågo04] Thomas Ågotnes. *A Logic of Finite Syntactic Epistemic States*. PhD thesis, Department of Informatics, University of Bergen, Norway, 2004.

[AML89] James S Albus, Harry G McCain, and Ronald Lumia. NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). NIST Technical Note 1235, National Institute of Standards and Technology, Robot Systems Division, Center for Manufacturing Engineering, Gaithersburg, MD 20899, 1989.

[Ark98] Ronald C Arkin. *Behaviour-Based Robotics*. MIT Press, 1998.

[Baj93] Ruzena Bajcsy, editor. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1993.

[BCPT03] Piergiorgio Bertoli, Alessandro Cimatti, Marco Pistore, and Paolo Traverso. A framework for planning with extended goals under partial observability. In *International Conference on Automated Planning and Scheduling*, pages 215–225, 2003.

[BCT04] Piergiorgio Bertoli, Alessandro Cimatti, and Paolo Traverso. Interleaving execution and planning for nondeterministic, partially observable domains. In *European Conference on Artificial Intelligence*, pages 657–661, 2004.

[Bee02] Michael Beetz. *Plan-Based Control of Robotic Agents: Improving the Capabilities of Autonomous Robots*, volume 2554 of *Lecture Notes in Computer Science*. Springer, 2002.

[BFT07] Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors. *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*. AAAI Press, 2007.

[BG01] Blai Bonet and Hector Geffner. Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence*, 14(3):237–252, 2001.

[BL97] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1–2):245–271, 1997.

[Bro91] Rodney A Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1–3):139–159, 1991.

[BS99] Liviu Badea and Monica Stanciu. Refinement operators can be (weakly) perfect. *Proceedings of the 9th International Workshop on Inductive Logic Programming*, 1634:21–33, 1999.

[CD97] Marco Cadoli and Francesco M Donini. A survey on knowledge compilation. *AI Communications*, 10(3–4):137–150, 1997.

[CM03] Simon Colton and Stephen Muggleton. ILP for mathematical discovery. In *13th International Conference on Inductive Logic Programming*, pages 93–111, 2003.

[COOP02] Waiyian Chong, Mike O'Donovan-Anderson, Yoshi Okamoto, and Don Perlis. Seven days in the life of a robotic agent. In *GSFC/JPL Workshop on Radical Agent Concepts*, pages 243–256, 2002.

[CRB04] Alessandro Cimatti, Marco Roveri, and Piergiorgio Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1–2):127–206, 2004.

[CS92] Marco Cadoli and Marco Schaerf. Approximate reasoning and non-omniscient agents. In *Proceedings of the 4th conference on Theoretical Aspects of Reasoning about Knowledge*, pages 169–183, 1992.

[CS01] Silvia Coradeschi and Alessandro Saffiotti. Perceptual anchoring of symbols for action. In *17th International Joint Conferences on*

Artificial Intelligence (IJCAI 2001), pages 407–412, Seattle, WA, 2001.

[DF95] Thomas G Dietterich and Nicholas S Flann. Explanation-based learning and reinforcement learning: A unified view. In *International Conference on Machine Learning*, pages 176–184, 1995.

[DRD01] Saso Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52, 2001.

[DRL96] Luc Dehaspe, Luc De Raedt, and Wim Van Laer. CLAUDIEN: the CLAUsal DIscovery ENgine user's guide, 1996.

[DW91] Thomas Dean and Michael P Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[Ebb99] Heinz-Dieter Ebbinghaus. Is there a logic for polynomial time? *Logic Journal of the IGPL*, 7(3):359–374, 1999.

[EKM+99] Jennifer Elgot-Drapkin, Sarit Kraus, Michael Miller, Madhura Nirkhe, and Don Perlis. Active logics: A unified formal approach to episodic reasoning. Technical Report CS-TR-4072, University of Maryland, 1999.

[Elg88] Jennifer Elgot-Drapkin. *Step Logic: Reasoning Situated in Time*. PhD thesis, Department of Computer Science, University of Maryland, 1988.

[Elg91] Jennifer Elgot-Drapkin. Step-logic and the three-wise-men problem. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI 1991)*, pages 412–417, 1991.

[FG00] Michael Fisher and Chiara Ghidini. Agents playing with dynamic resource bounds. In Markus Hannebauer, editor, *Proceedings of the Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems (ECAI 2000)*, 2000.

[FHVM95] Ronald Fagin, Joseph Y Halpern, Moshe Y Vardi, and Yoram Moses. *Reasoning about knowledge*. MIT Press, 1995.

[Fir89] R James Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Department of Computer Science, Yale University, 1989.

[FS87] Kenneth Forbus and Howard E Shrobe, editors. *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI 1987)*. MIT Press, 1987.

[FYG04] Alan Fern, SungWook Yoon, and Robert Givan. Learning domain-specific control knowledge from random walks. In *International Conference on Automated Planning and Scheduling*, pages 191–199, 2004.

[Gat91] Eran Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Virginia Politechnic Institute and State University, 1991.

[GINR99] Giuseppe De Giacomo, Luca Iochhi, Daniele Nardi, and Riccardo Rosati. A theory and implementation of cognitive mobile robots. *Journal of Logic and Computation*, 9(5):759–785, 1999.

[GKP00] John Grant, Sarit Kraus, and Don Perlis. A logic for characterizing multiple bounded agents. *Autonomous Agents and Multi-Agent Systems*, 3(4):351–387, 2000.

[GL87] Michael P Georgeff and Amy L Lansky. Reactive reasoning and planning. In Kenneth Forbus and Howard E Shrobe, editors, *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI 1987)*, pages 677–682, 1987.

[GLP05] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[GPS98] Goran Gogic, Christos H Papadimitriou, and Martha Sideri. Incremental recompilation of knowledge. *Journal of Artificial Intelligence Research*, 8:23–37, 1998.

[GT04] Charles Gretton and Sylvie Thiebaux. Exploiting first-order regression in inductive policy selection. In *Conference on Uncertainty in Artificial Intelligence*, pages 217–225, 2004.

[GVB04] Christophe Giraud-Carrier, Ricardo Vilalta, and Pavel Brazdil. Introduction to the special issue on meta-learning. *Machine Learning*, 54(3):187–193, 2004.

[GW01] Dov Gabbay and John Woods. The new logic. *Logic Journal of the IGPL*, 9(2):141–174, 2001.

[HBHM99] Koen V Hindriks, Frank S de Boer, Wiebe van der Hoek, and John-Jules Ch Meyer. Control structures of rule-based agent languages. In Müller et al. [MSR99], pages 381–396.

[HHM+00] Henry Hexmoor, Marcus Huber, Jörg P Müller, John Pollock, and Donald Steiner. On the evaluation of agent architectures. In Nicholas R Jennings and Yves Lespérance, editors, *Proceedings of 6th International Workshop Intelligent Agents VI: Agent Theories, Architectures and Languages (ATAL 1999)*, volume 1757 of *Lecture Notes in Computer Science*, pages 106–116. Springer, 2000.

[HPS04] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[HW02] Wiebe van der Hoek and Michael Wooldridge. Tractable multiagent planning for epistemic goals. In *First International Conference on Autonomous Agents and Multiagent Systems*, pages 1167–1174, 2002.

[HW03] Wiebe van der Hoek and Michael Wooldridge. Cooperation, knowledge and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75:125–157, 2003.

[JSW98] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[KBM98] David Kortenkamp, R Peter Bonasso, and Robin Murphy, editors. *Artificial Intelligence and Mobile Robots*. AAAI Press / MIT Press, 1998.

[Kha99] Roni Khardon. Learning to take actions. *Machine Learning*, 35(1):57–90, 1999.

[KL06] Tolga Könik and John E Laird. Learning goal hierarchies from structured observations and expert annotations. *Machine Learning*, 64:263–287, 2006.

[Koe01] Sven Koenig. Agent-centered search. *Artificial Intelligence Magazine*, 4(22):109–131, 2001.

[KR95] Roni Khardon and Dan Roth. Learning to reason with a restricted view. In *Workshop on Computational Learning Theory*, pages 301–310, 1995.

[KR97] Roni Khardon and Dan Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, 1997.

[LC06] Pat Langley and Dongkyu Choi. Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, 7:493–518, 2006.

[Lee00] Jaeho Lee. Reactive-system approaches to agent architectures. In Nicholas R Jennings and Yves Lespérance, editors, *Proceedings of 6th International Workshop Intelligent Agents VI: Agent Theories, Architectures and Languages (ATAL 1999)*, volume 1757 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2000.

[Lev84] Hector Levesque. A logic of implicit and explicit belief. In *Proceedings of 4th National Conference on Artificial Intelligence (AAAI 1984)*, pages 198–202, 1984.

[Lew01] Richard L Lewis. Cognitive theory, SOAR. *International Encyclopedia of the Social and Behavioral Sciences*, pages 2178–2183, 2001.

[LK02] Maxim Likhachev and Sven Koenig. Lifelong planning for mobile robots. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents*, pages 140–156. Springer-Verlag, 2002.

[LSBM06] Derek Long, Stephen F Smith, Daniel Borrajo, and Lee McCluskey, editors. *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006)*. AAAI Press, 2006.

[McD92] Drew McDermott. Transformational planning of reactive behavior. Technical Report TR CSD RR-941, Yale University, 1992.

[Mit97] Thomas M Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[Mor04] Eduardo F Morales. Relational state abstractions for reinforcement learning. In *Proceedings of the Workshop on Relational Reinforcement Learning (ICML 2004)*, pages 27–32, 2004.

[Moy02] Stephen Moyle. Using theory completion to learn a robot navigation control program. In *Inductive Logic Programming*, pages 182–197, 2002.

[MPT95] Jörg P Müller, Markus Pischel, and Michael Thiel. A pragmatic approach to modeling autonomous interacting systems – preliminary report. In Michael Wooldridge and Nicholas R Jennings, editors, *Proceedings of Workshop on Agent Theories, Architectures, and Languages (ECAI 1994)*, volume 890 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 1995.

[MSR99] Jörg P Müller, Munindar P Singh, and Anand S Rao, editors. *Proceedings of 5th International Workshop Intelligent Agents V: Agent Theories, Architectures and Languages (ATAL 1998)*, volume 1555 of *Lecture Notes in Computer Science*. Springer, 1999.

[Mug95] Stephen Muggleton. Inverse entailment and PROGOL. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3–4):245–286, 1995.

[Mül99] Jörg P Müller. The right agent (architecture) to do the right thing. In Jörg P Müller, Munindar P Singh, and Anand S Rao, editors, *Proceedings of 5th International Workshop Intelligent Agents V: Agent Theories, Architectures and Languages (ATAL 1998)*, volume 1555 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 1999.

[NKP93] Madhura Nirkhe, Sarit Kraus, and Don Perlis. Situated reasoning within tight deadlines and realistic space and computation bounds. In *Proceedings of the 2nd Symposium on Logical Formalizations of Commonsense Reasoning*, 1993.

[NM07a] Sławomir Nowaczyk and Jacek Malec. An architecture for resource bounded agents. In *Workshop on Agent Based Computing (ABC'07)*, Wisła, Poland, 2007.

[NM07b] Sławomir Nowaczyk and Jacek Malec. Inductive logic programming algorithm for estimating quality of partial plans. In *Proceedings of the 6th Mexican International Conference on Artificial Intelligence*, volume 4827 of *Lecture Notes in Computer Science*, pages 359–369. Springer, 2007.

[NM07c] Sławomir Nowaczyk and Jacek Malec. Learning to evaluate conditional partial plans. In *ICMLA '07: Proceedings of the Sixth International Conference on Machine Learning and Applications*, pages 235–240. IEEE Computer Society, 2007.

[NM07d] Sławomir Nowaczyk and Jacek Malec. Relative relevance of subsets of agent's knowledge. In *Workshop on Logics for Resource Bounded Agents*, September 2007.

[Now06a] Sławomir Nowaczyk. Learning of agents with limited resources. In *AAAI-06 Student Abstract and Poster Program*, 2006.

[Now06b] Sławomir Nowaczyk. Partial planning for situated agents based on active logic. In *Workshop on Logics for Resource Bounded Agents (ESSLLI 2006)*, 2006.

[Nyb05] Per Nyblom. Handling uncertainty by interleaving cost-aware classical planning with execution. In *3rd joint SAIS-SSL event on Artificial Intelligence and Learning Systems*, pages 134–140, 2005.

[Pat85] Peter F Patel-Schneider. A decidable first-order logic for knowledge representation. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI 1985)*, pages 455–458, 1985.

[Pat86] Peter F Patel-Schneider. A four-valued semantics for frame-based description languages. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986)*, pages 344–348, 1986.

[PB04] Ronald P A Petrick and Fahiem Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 2–11, 2004.

[PPT+99] Khemdut Purang, Darsana Purushothaman, David Traum, Carl Andersen, and Don Perlis. Practical reasoning and plan execution with active logic. In John Bell, editor, *Proceedings of the Workshop on Practical Reasoning and Rationality (IJCAI 1999)*, pages 30–38, 1999.

[Rei01] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[RG93] Anand S Rao and Michael P Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI 1993)*, pages 318–324, 1993.

[RMM+94] Francesco Ricci, S Mam, Patrizia Marti, Véronique Normand, and Pilar Olmo. CHARADE: a platform for emergencies management systems. Technical Report 9404-07, Trento University, Povo, Italy, 1994.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[SB98] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998. A Bradford Book.

[SEKW] Edgar Sommer, Werner Emde, Jörg-Uwe Kietz, and Stefan Wrobel. MOBAL 3.0 user guide.

[SK96] Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

[Sut90] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Bruce W Porter and Raymond J Mooney, editors, *Proceedings of the 7th International Conference on Machine Learning (ICMLA 1990)*, pages 216–224. Morgan Kaufmann Publishers, Inc., 1990.

[VD02] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[WJ95] Michael Wooldridge and Nicholas R Jennings, editors. *Proceedings of Workshop on Agent Theories, Architectures, and Languages (ECAI 1994)*, volume 890 of *Lecture Notes in Computer Science*. Springer, 1995.

[WL01] Michael Wooldridge and Alessio Lomuscio. A computationally grounded logic of visibility, perception, and knowledge. *Logic Journal of the IGPL*, 9(2):257–272, 2001.

[Wob00] Wayne Wobcke. On the correctness of PRS agent programs. In Nicholas R Jennings and Yves Lespérance, editors, *Proceedings of 6th International Workshop Intelligent Agents VI: Agent Theories, Architectures and Languages (ATAL 1999)*, volume 1757 of *Lecture Notes in Computer Science*, pages 42–56. Springer, 2000.

[Woo00] Michael Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[WR99] Michael Wooldridge and Anand Rao, editors. *Foundations of Rational Agency*. Kluwer Academic Publishers, 1999.

[Yam96] Akihiro Yamamoto. Improving theories for inductive logic programming systems with ground reduced programs. Technical report, AIDA9619, Technische Hochschule Darmstadt, 1996.

[YZ92] Ronald R Yager and Lofti A Zadeh, editors. *An Introduction to Fuzzy Logic Applications in Intelligent Systems*. Springer Media, 1992.

[ZR93] Shlomo Zilberstein and Stuart J Russell. Anytime sensing, planning and action: A practical model for robot control. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1401–1407. Morgan Kaufmann, 1993.