Partial Planning for Situated Agents based on Active Logic

Sławomir Nowaczyk
Slawomir.Nowaczyk@cs.lth.se
Department of Computer Science
Lund University, Sweden

Abstract

This paper presents an investigation of rational agents that have limited computational resources and intentionally interact with their environments. We present an example logical formalism, based on Active Logic and Situation Calculus, that can be employed in order to satisfy the requirements arising due to being situated in a dynamic universe. We analyse how such agents can combine, in a time-aware fashion, inductive learning from experience and deductive reasoning using domain knowledge. In particular, we consider how partial plans are created and reasoned about, focusing on what new information can be provided as a result of action execution.

1 Introduction

In our research we are interested in building rational agents that can interact with their environment. In order to be practically useful, such agents should be modelled as having bounded computational resources. Moreover, since they are situated in a dynamic world, they need to be aware of the notion of time — in particular, that their reasoning process is not instantaneous. On the other hand, such agents have the possibility to acquire important knowledge by observing the environment surrounding them and by analysing their past interactions with it.

This paper mainly focuses on presentation of one kind of logical formalism we think may be appropriate for such agents. We describe how Active Logic can be augmented with epistemic concepts and combined with mechanism related to Situation Calculus, in order to provide flexible and efficient reasoning formalism for rational agents.

We also present how such agents can deal with planning in domains where complexity makes finding complete solutions intractable. Clearly, it is often not realistic to expect an agent to be able to find a total plan which solves a problem at hand. Therefore, we investigate how an agent can create and reason about *partial plans*. By that we mean plans which bring it somewhat closer to achieving the goal, while still being simple and short enough to be computable in reasonable time. Currently we mainly concentrate on plans which allow an agent to acquire additional knowledge about the world.

By executing such “information-providing” partial plans, an agent can greatly simplify subsequent planning process — it no longer needs to take into account the vast number of possible situations which will be inconsistent with 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.

If the environment is modelled sufficiently well (for example, if a simulator exists), the agent may have a high degree of freedom in exploring it and in deciding how to interact with it. It may be possible to gain information that the agent would not be able to, by itself, observe directly. In many domains it is significantly easier to build and employ a simulator than to analytically predict results of complex interactions. In other cases, for example when the agent is a robot situated in an unknown environment, it must learn “in the wild” and be aware that the actions it executes are final: they do happen and there is no way of undoing them, other than performing, if possible, a reverse action.

In order to accommodate all of the above we use a variant of Active Logic (Elgot-Drapkin *et al.* 1999) as agent’s reasoning formalism. It was designed for non-omniscient agents and has mechanisms for dealing with uncertain and contradictory knowledge. We believe that Active Logic is a good reasoning technique for versatile agents, in particular as it has been applied successfully to several different problem domains, including some in which planning plays a very prominent role (Purang *et al.* 1999). Moreover, in order to be able to intentionally direct its own learning process, the agent needs to reason about its own knowledge and lack of thereof — thus, the logic we use has been augmented with epistemic concepts (Fagin *et al.* 1995).

In other words, our agents are supposed to combine deductive and inductive reasoning with time-awareness. We believe that the interactions among those three aspects are crucial for developing truly intelligent systems. It is not our goal to analyse strict deadlines or precise time measurements (although we do not exclude a possibility of doing that), but rather to express that a rational agent needs the ability to reason about committing its resources to various tasks (Chong *et al.* 2002). In particular, it is not justified to assume that an agent knows all deductive consequences of its own beliefs.

2 Wumpus Game

The example problem we will be using through this paper is a well-known game of Wumpus, a classic testbed for intelligent agents. In its basic form, the game takes place on a rectangular board through which an player is allowed to move freely. A beast called Wumpus occupies one, initially unknown, square. Agent’s goal is to kill the creature, a task that can be achieved by shooting an arrow on that square. Luckily, Wumpus is a smelly creature, so the player always knows if the monster is nearby. But unfortunately, not the exact direction to it. At the same time, when walking around, the player needs to avoid stumbling across the monster, or else he

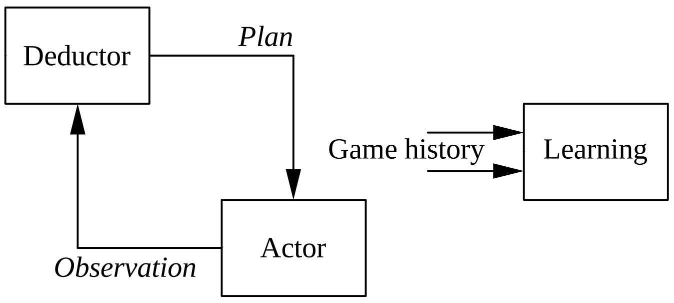

Figure 1: The architecture of the system.

gets eaten by it.

This game is concise enough to be explained easily, but finding a solution is sufficiently complex to illustrate the issues we want to emphasise. We look at it as one instance of a significantly broader class of problems, along the lines of *General Game Playing*, where an agent accepts a formal description of an arbitrary game and, without further human interaction, can play it effectively.

3 Agent Architecture

We use a simple architecture for our agent, as presented in Fig. 1. It consists of three main elements, corresponding to the three main tasks of the agent.

The Deductor reasons about world, possible actions and what could be their consequences. Its main aim is to generate plans applicable in current situation and predict — at least as far as past experience, imperfect domain knowledge and limited computational resources allow — effects each of them will have, in particular what new knowledge can be acquired.

The Actor is responsible for overseeing the reasoning process, mainly for introducing new observations into the knowledge base and for choosing plans for execution. Basically, it decides *when* to switch from deliberation to acting, and which of the plans under consideration to execute.

These two modules form the core of the agent. By creating and executing a sequence of partial plans our agent moves progressively closer and closer to its goal, until it reaches a point where a winning plan can be directly created by Deductor, and its correctness can be proven.

The learning module is necessary in order to ensure that the plans agent chooses for execution are indeed “good” ones. After the game is over, regardless of whether the agent has won or lost, learning system inductively generalises experience it has gathered — attempting to improve Deductor’s and Actor’s performance. Our goal is 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 select the best partial plan.

4 Deductor

In this section we first explain the knowledge representation used, then point out some of the more interesting issues with the reasoning machinery, and finally we introduce a simplified example illustrating how our example Wumpus domain could be axiomatised.

4.1 Knowledge Representation

The language used by Deductor is the First Order Logic augmented with some mechanisms similar to those of Situation Calculus. Within a given situation, knowledge is expressed using standard FOL. In particular, we do not put any limitations on the expressiveness of the language. Predicate $Knows$ describes knowledge of the agent, e.g.,

$$Knows[\ Smell(a) \leftrightarrow \exists_x Wumpus(x) \land Neighbour(a,x)\ ]$$

means: *agent knows that it smells on exactly those squares which neighbour Wumpus' position.* The predicate $Knows$ may be nested, a feature which is very useful, if only in a number of specialised contexts — mainly for allowing an agent to reason about future and about what effects performed actions will have. We use standard reification mechanism for putting formulae as parameters of a predicate (Reiter 2001).

In order to describe actions and change, we employ a variant of well-known Situation Calculus approach (McCarthy & Hayes 1969), except that we use predicate $Knows$ instead of $Holds$ — in order to make it explicit that out main priority is describing agent's knowledge. There are some semantic differences between our $Knows$ and a typical $Holds$, however — for example,

$$Knows(s, \alpha \lor \beta) \rightarrow Knows(s, \alpha) \lor Knows(s, \beta)$$

is *not* a tautology, while a corresponding formula for $Holds$ usually is. We will discuss these issues later in this section.

Nevertheless, we introduce an additional parameter to the $Knows$ predicate, which denotes the current situation. Moreover, since the agent is required to reason about knowledge-producing actions, we add yet another parameter, namely the plan agent is going to execute.

In other words, the first formula in this section should be more properly written in the form:

$$Knows[s, p,\ Smell(a) \leftrightarrow \exists_x (Wumpus(x) \land Neighbour(a,x))\ ]$$

and mean: *agent knows that executing plan p in situation s leads to a new situation, such that it smells on exactly those squares which neighbour Wumpus' position.* This particular formula is an universal law of the world, valid regardless of the chosen s and p, but many interesting ones — e.g. "$Wumpus(a)$" or "$Knows[Smell(b)]$" — are true only for specific s and p.

The initial knowledge of the agent, one concerning current situation, represents the state of the world after execution of the empty plan. From this, using typical STRIPS-like representation of actions (i.e. preconditions and effects), the agent can reason about extending a particular plan by various operations. Next, for every plan obtained this way, it can deduce which formulae are valid in situation(s) resulting from its execution.

One important extension to the classical STRIPS formalism we employ is support for conditional actions. Such actions are important when some of agent's actions may provide information which is necessary to carry on. Basically, the plans agent reasons about consist of a concatenation of classical and conditional actions, the latter of the form $(predicate\ ?\ action_1 : action_2)$. Those have the standard meaning, i.e. that $action_1$ will be executed if $predicate$ holds, and $action_2$ will be executed otherwise. For a well-developed discussion of other possible ways of representing conditional partial plans and of interleaving planning and execution see, for example, Bertoli, Cimatti, & Traverso (2004).

Conceptually, the agent creates new plans by inductively extending each and every plan considered up to now by each and every possible action. Obviously, many plans created this way would be either invalid or clearly uninteresting. Therefore, due to the computational complexity issues of such a naive approach, we implement plan creation and validation as a process external to the logical reasoner. In general, the main requirement is to ensure that each plan an agent reasons about is valid, i.e. that the preconditions of each action are fulfilled.

We use standard Situation Calculus representation of the actions agent can execute, i.e. using pre– and postconditions. Since some of those actions can be knowledge-producing, it is important to represent that fact properly in their effects. For example, moving onto square a will make an agent know whether $Smell(a)$ is true or false. Of course, this information will only be available during plan execution, not during the planning itself. But it is necessary to distinguish that agent *will have* this knowledge, so that it is able to create a conditional plan branching on value of this predicate. In order to properly represent this notion, we introduce a predicate $KnowsIf$, syntactically similar to $Knows$, except that the meaning of $KnowsIf(s, p, \alpha)$ is that *agent knows that executing plan p in situation s leads to a new situation, in which it* will know *whether α is true or false — but it does not necessarily know, at planning time, which one.*

Extra care needs to be taken when creating conditional plans, as it is important to make sure that the agent has, during plan execution, enough knowledge to correctly choose the appropriate conditional branch. Basically, a precondition for a conditional action $(predicate\ ?\ action_1 : action_2)$ is the conjunction of preconditions for $action_1$, preconditions for $action_2$ and truthfulness of $KnowsIf(predicate)$ — the fact that agent knows whether $predicate$ holds or not.

Finally, due to semantical differences between $Holds$ and $Knows$ predicates, we have found it advantageous to introduce an explicit distinction between *fluents* and *domain constraints*. Semantically, the difference is that truth value of fluents depends on current situation, while the truth value of domain constraints remains fixed for the duration of game episode (although *agent's knowledge* of them can, obviously, change).

To this end we introduce a predicate $Invariant$, distinguishing formulae which are independent of current situation, and a special inference rule which allows agent to propagate knowledge of domain constraints from one situation to another.

4.2 Reasoning

We will start this subsection by describing the reasoning process *within* a given situation, i.e. while an agent ponders which actions to execute. Later we will shortly mention problems that arise when an action is executed and the state of the world changes, and we will conclude with some reference to reasoning about *different game episodes*, which is still very much work in progress.

As we said previously, an agent will create a number of plans and each such plan will be evaluated by an Actor. Therefore, Deductor can devote more effort to plans that are more promising. Since the agent employs Active Logic, this can be easily modelled within that formalism, as it is intended to describe the deduction as an ongoing process, rather than characterising only some static, fixed-point consequence relation.

Active Logic annotates every formula with a time-stamp (usually an integer) of when it was first derived, incrementing the label with every application of an inference rule:

$$\frac{i:\quad a, a \to b}{i+1:\quad b}$$

It also includes the Now predicate, true only during current time point (i.e., "$i : Now(j)$" is true for all $i = j$, but false for all $i \neq j$). It can, therefore, use this time-stamp to prioritise plans. For example, if an Actor divides the plans into two classes, *normal* ones and *great* ones, we can have inference rules of the kind:

$$\frac{i:\quad Great(s,P), Knows(s,P,a), Knows(s,P,a \to b)}{i+1:\quad Knows(s,P,b)}$$

$$\frac{i:\quad Now(i), even(i), Knows(s,P,a), Knows(s,P,a \to b)}{i+1:\quad Knows(s,P,b)}$$

which would ensure that reasoning about *great* plans happens at every step, but reasoning about *normal* ones only happens every other step. This is a very simple example, more complex scenarios are certainly possible, but their usefulness has to be evaluated experimentally. In particular, we find the idea of allowing *the agent* to consciously balance this tradeoff very stimulating.

In a similar spirit, we can balance the tradeoff between creating more complex, longer plans and reasoning about effects of already created plans. At this point it is, however, somewhat unclear on what basis should the agent make decisions regarding this. Nevertheless, another feature of Active Logic, namely the *observation* *function*, which delivers axioms that are valid since a specific point in time, can be used quite naturally to acquire new plans — possibly from a module external to the reasoner itself.

We mentioned above that the $Invariant$ predicate requires a specialised inference rule. There are several possibilities, one being:

$$\frac{i : Knows(s, p, \alpha) \wedge Invariant(\alpha)}{i + 1 : Knows(s', p', \alpha)}$$

for every s, p, s', p'. In practice, the reasoner would not need to multiply the formulae for every possible combination of plan and situation, it can simply mark them as being invariant and take this into account during inference in an efficient way.

After the reasoning and planning has progressed sufficiently, the Actor will choose a plan and execute it. At that point the reasoner can discard other plans it has created — they are no longer needed — and needs to adapt to the new state of affairs. First, it needs to notice that current situation has changed — we use a predicate $State$, modelled closely after standard Active Logic predicate Now, in order to achieve this. And second, it needs to absorb the new knowledge acquired by the executing the actions, which can be done using observation function.

An interesting issue is also how to allow an agent to reason about past game episodes. In particular, after the game is won or lost and a new one is being started, a lot of knowledge acquired previously still remains valid and interesting. We are working on ways to extract general useful knowledge and to discover similarities in episodes. This is also closely related to the learning module, discussed in more detail in subsequent sections.

4.3 Wumpus Example

In this section we briefly introduce our example domain, the game of Wumpus. We use a rather natural set of axioms to describe it. First we specify that Wumpus is on exactly one square:

$$\exists_x \, Wumpus(x)$$

$$\forall_{x,y} \;\; x \neq y \rightarrow \neg Wumpus(x) \vee \neg Wumpus(y)$$

We define the smelling phenomenon:

$$\forall_x \, Smell(x) \leftrightarrow \exists_y \, Wumpus(y) \wedge Neighbour(x, y)$$

and that the agent will always know whether it smells on the square it is on:

$$\forall_x \, Player(x) \leftrightarrow \mathit{KnowsIf}(Smell(x))$$

We also define $Neighbour$ relation and other geometrical knowledge in a natural way.

Finally, we need to specify that $Wumpus$ remains stationary thorough the episode. This is the main reason for introduction of the $Invariant$ predicate in previous section. It allows us to state in a simple way that predicates such as $Wumpus, Neighbour, up, down, left, right, even$ etc. do not depend on the current situation. Therefore, any formula α containing only those predicates is an invariant. So, if an agent discovers in some situation s_1 that $Wumpus(a) \vee Wumpus(b)$, it can easily deduce that $Wumpus(a) \vee Wumpus(b)$ also holds in any other situation s_2.

Observe that a, somewhat more natural, rule such as:

$$\forall_{x,s,s',p,p'} Knows(s, p, Wumpus(x)) \leftrightarrow Knows(s', p', Wumpus(x))$$

does not quite work, as agent's knowledge is typically of the kind

$$Knows(s, p, Wumpus(a) \vee Wumpus(b))$$

and it is not clear how to propagate such formulae between situations without exponential blowup of axioms (problems arise, for example, from the fact that $Knows$ is not distributive to disjunction).

We currently perform grounding of all the variables in order to have reasoning in predicate logic.

4.4 Summary

In general, our agent reasons on three distinct levels of abstraction. First is *within* a given situation, where it tries to find the best plan to execute. Second level concerns effects of executed actions, where state of the world changes and new knowledge becomes available. And third deals with comparing different game episodes, where knowledge previously assumed to be $Invariant$ is no longer so (for example, Wumpus may now be at another location).

The Active Logic formalism used, especially predicate Now, makes it possible to reason about passing time and, combined with observation function delivering knowledge about external events, allows the agent to remain responsive during its deliberations. This way both Deductor and Actor can keep track of how the reasoning is progressing and make informed decisions about balancing thinking and acting.

One of the reasons we have chosen symbolic representation of plans, as opposed to a policy (an assignment of 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 Bertoli *et al.* (2003), where authors present a solution for planning with goals described in Computational Tree Logic. This formalism allows to express goals of the kind "value of *a* will never be changed", "*a* will be eventually restored to its original value" or "value of *a*, after time *t*, will always be *b*" etc.

To summarise, our agent uses Active Logic to reason about its own knowledge, which is very important in the Wumpus domain. Here, the main goal can be reduced to "learn the position of Wumpus", so active planning for knowledge acquisition is crucial. Agent also requires an ability to compare what kind of information will execution of each plan provide, in order to be able to choose the best one of them.

5 Actor

The Actor module supervises the deduction process and breaks it at selected moments, e.g., when it notices a particularly interesting plan or when it decides that sufficiently long time has been spent on planning. It then *evaluates* existing partial plans and executes the best one of them. The evaluation process is crucial here, and we expect the subsequent learning process to greatly contribute to its improvement. In the beginning, the choice may be done at random, or some simple heuristic may be used. After execution of partial plan, a new situation is reached and the Actor lets the Deductor create another set of possible plans.

This is repeated as many times as needed, until the game episode is either won or lost. Losing the game clearly identifies bad choices on the part of the Actor and leads to an update of the evaluation function.

Winning the game also yields feedback that may be used for improving this function, but it also provides a possibility to (re)construct a complete plan, i.e. one which starts in the initial situation and ends in a winning state. If such a plan can be found, it may be subsequently used to quickly solve any problem instance for which it is applicable. Moreover, even if such plan is not directly applicable, an Actor can use it when evaluating other plans found by the Deductor. Those with structure similar to the successful one are more likely to be worthwhile.

6 Learning

When analysing learning module, it is important to keep in mind that our agent has a dual aim, akin to the exploration and exploitation dilemma in reinforcement learning. On one hand, it wants to win the current game episode, but at the same time it needs to learn as much general knowledge as possible, in order to improve its future performance.

Currently we are mainly investigating the learning module from Actor's perspective — using ILP to evaluate quality of partial plans is, to the best of our knowledge, a novel idea. One issue is that work on ILP has been dealing almost exclusively with the problem of *classification*, while our situation requires *evaluation*. 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.

For now, however, we focus on distinguishing a special class of "bad" plans, namely ones that lead to losing the game. Clearly some plans — those that in agent's experience *did* so — are bad ones. But not every plan which does not cause the agent to lose is a *good* plan. Further, not every plan that leads to *winning* a game is a good one. An agent might have executed a dangerous plan and win only because it has been lucky.

Therefore, we define as positive examples those plans which lead, or can be proven to *possibly* lead, to the defeat. On the other hand, those plans which can be proven to *never* cause defeat are negative examples. There is a third class of plans, when neither of the above assertions can be proven. We are working on how to use such examples in learning most effectively.

Nevertheless, this is only the beginning. After all, in many situations a more "proactive" approach than simple *not-losing* is required. One promising idea is to explore the epistemic quality of plans: an agent should pursue those which provide the most important knowledge. Another way of expressing distinction between good and bad partial plans, one we feel can give very good results, is discovering relevant subgoals and landmarks, as in Hoffmann, Porteous, & Sebastia (2004).

7 Environment Interaction

Another interesting issue in our framework is 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, an 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 such reasoning, our agent requires an ability to both act in the world and to observe it. Finally, it needs to consider 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 different models of interactions with the world are possible.

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, a physical modelling, where it is often much easier to simulate things than to predict their behaviour and interactions. In a similar spirit, it may be easier for our agent to "ask the environment" about validity of some formula than to prove it.

If 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. Alternatively, it may store some opaque *situation identifier* so that it can revisit the same situation at later time. This model is also suitable for agents that do not have perfect knowledge of the world, as the "replay" capability does not assume *the agent* is able to fully reconstruct the situation or knows the state of the world completely.

In our opinion, this is the most interesting setting: it gives the agent sufficient freedom to allow it to achieve interesting results and at the same time is not, in many domains, overly infeasible. On the other hand, we are working on ways in which this setting could be made even more practical — one idea is having an agent accept the fact that in several replays "the same" situation could vary slightly. For example, physical agent might request an operator to restore the previous state of the world: it would not really be identical, but it may be sufficiently close. Alternatively, in some application domains, only a subset of situations may be "replayable" — only those, for example, that an agent can restore, with required tolerance, all by itself.

In most applications, however, the agent is only able to influence its own actions and have no control whatsoever over the rest of the world. This is also the most suitable model for an *autonomous* physical agent. In such 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. It thus needs to consider if saving some deduction effort is indeed the best possible course of action, or if doing something else instead would be more advantageous.

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 needs to, first, assert that an action is reasonably safe. In this case, unlike the previous one, 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 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 agent free to ponder more "philosophical" issues. On the other hand, when the ball is rolling in agent's direction, time is of essence and an agent better had plans ready for several most plausible action outcomes.

8 Related Work

Combination of planning and learning is an area of active research, in addition to the extensive amount of work being done separately in those respective fields.

There has been significant amount of work done in learning about what actions to take in a particular situation. One notable example is Khardon (1999), where author showed important theoretical results about PAC-learnability of action strategies in various models. In Moyle (2002) author discussed a more practical approach to learning Event Calculus programs using Theory Completion. He used extraction-case abduction and the ALECTO system in order to simultaneously learn two mutually related predicates ($Initiates$ and $Terminates$) from positive-only observations. Recently, Könik & Laird (2004) developed a system which is able to learn low-level actions and plans from goal hierarchies and action examples provided by experts, within the SOAR architecture.

The work mentioned above focuses primarily on learning how to act, without focusing on reaching conclusions in a deductive way. In a sense, the results are somewhat more similar to the reactive-like behaviour than to classical planning system, with important similarities to the reinforcement learning and related techniques.

One attempt to escape the trap of large search space has been presented in

Džeroski, Raedt, & Driessens (2001), 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 actually being learned via behaviour cloning techniques, is presented in Morales (2004).

Recently, Colton & Muggleton (2003) showed several ideas about 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 (Fern, Yoon, & Givan 2004), where the system learns domain-dependent control knowledge beneficial in planning tasks.

Yet another track of research focuses on (deductive) planning, taking into account incompleteness of agent's knowledge and uncertainty about the world. Conditional plans, generalised policies, conformant plans, universal plans and some others are the terms used by various researchers (Cimatti, Roveri, & Bertoli 2004; Bertoli, Cimatti, & Traverso 2004) to denote in principle the same 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 Bonet & Geffner (2001) or earlier in Dean & Wellman (1991). We are not aware of any such research that would attempt to integrate learning.

9 Conclusions

The work presented here is still very much in progress and a discussion of an interesting track of research, rather than a report on some concrete results. We have introduced an agent architecture facilitating resource-aware deductive planning interwoven with plan execution and supported by inductive, life-long learning. The particular deduction mechanism used is based on Active Logic, in order to incorporate time-awareness into the reasoning itself. The plans created in deductive way are conditional, accounting for possible results of future actions, in particular information-gathering ones.

We intend to continue this work in several directions. Discovering subgoals and subplans seems to be one of the most useful capabilities of human problem solving and we would like our agent to invent and use such concept. In our example domain a useful subgoal could be "First, find a place where it smells." In addition, Deductor should be able to conceive general rules of rational behaviour, such as "Don't shoot if you don't know Wumpus' position". Yet another clear advantage would be the ability to reuse a previously successful plan in a different situation. Finally, domain experts often are an invaluable source of knowledge that the agent should be able to exploit, if possible.

The ideas above do not cover all the possible further investigations and extensions of the proposed system; it is just a biased presentation of the authors' own interests and judgements.

References

Bertoli, P.; Cimatti, A.; Pistore, M.; and Traverso, P. 2003. A framework for planning with extended goals under partial observability. In *International Conference on Automated Planning and Scheduling*, 215–225.

Bertoli, P.; Cimatti, A.; and Traverso, P. 2004. Interleaving execution and planning for nondeterministic, partially observable domains. In *European Conference on Artificial Intelligence*, 657–661.

Bonet, B., and Geffner, H. 2001. Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence* 14(3):237–252.

Chong, W.; O'Donovan-Anderson, M.; Okamoto, Y.; and Perlis, D. 2002. Seven days in the life of a robotic agent. In *GSFC/JPL Workshop on Radical Agent Concepts*.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159(1-2):127–206.

Colton, S., and Muggleton, S. 2003. ILP for mathematical discovery. In *13th International Conference on Inductive Logic Programming*.

Dean, T., and Wellman, M. P. 1991. *Planning and Control*. Morgan Kaufmann.

Džeroski, S.; Raedt, L. D.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43(1/2):7–52.

Elgot-Drapkin, J.; Kraus, S.; Miller, M.; Nirkhe, M.; and Perlis, D. 1999. Active logics: A unified formal approach to episodic reasoning. Technical Report CS-TR-4072, University of Maryland.

Fagin, R.; Halpern, J. Y.; Vardi, M. Y.; and Moses, Y. 1995. *Reasoning about knowledge*. MIT Press.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *International Conference on Automated Planning and Scheduling*.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Khardon, R. 1999. Learning to take actions. *Machine Learning* 35:57–90.

Könik, T., and Laird, J. 2004. Learning goal hierarchies from structured observations and expert annotations. In *14th International Conference on Inductive Logic Programming*.

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 4:463–502.

Morales, E. P. 2004. Relational state abstraction for reinforcement learning. In *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning*.

Moyle, S. 2002. Using theory completion to learn a robot navigation control program. In *12th International Conference on Inductive Logic Programming*.

Purang, K.; Purushothaman, D.; Traum, D.; Andersen, C.; and Perlis, D. 1999. Practical reasoning and plan execution with active logic. In *Proceedings of the IJCAI-99 Workshop on Practical Reasoning and Rationality*, 30–38.

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