Learning to Evaluate Conditional Partial Plans

Sławomir Nowaczyk and Jacek Malec

Department of Computer Science, Lund University
Slawomir.Nowaczyk@cs.lth.se
Jacek.Malec@cs.lth.se

Abstract. We study agents situated in partially observable environments, who do not have sufficient resources to create conformant (complete) plans. Instead, they create plans which are conditional and partial, execute or simulate them, and learn from experience to evaluate their quality. Our agent employs an incomplete symbolic deduction system based on Active Logic and Situation Calculus for reasoning about actions and their consequences. An Inductive Logic Programming algorithm generalises observations and deduced knowledge so that the agent can choose the best plan for execution.
We show results of using PROGOL learning algorithm to distinguish "bad" plans early in the reasoning process, before too many resources are wasted on considering them. We show that additional knowledge needs to be provided before learning can be successful, but argue that the benefits achieved make it well worth the additional effort.
Finally, we identify several assumptions made by PROGOL, shared by other similarly universal algorithms, which — while well justified in general — fail to exploit the properties of some class of problems faced by rational agents.[1]

1 Introduction

Rational, autonomous agents able to survive and achieve their goals in dynamic, only partially observable environments are the ultimate dream of AI research since its beginning. Quite a lot has already been done towards achieving that dream, but dynamic environments still are a big challenge for autonomous systems. In particular, nontrivial environments that are only partially observable pose demands which are beyond the current state of the art, possibly except when dedicated solutions are developed for specific domains.

One of the major ways of coping with uncertainty and lack of knowledge about current situation is to exploit previous experience. In our research we are interested in developing rational, situated agents that are aware of their own limitations and can take them into account, as brilliantly presented by Chong and others in [1].

Due to limited resources and the necessity to stay responsive in a dynamic world, situated agents cannot be expected to create a complete plan for achieving their goals. A common approach is to create a *conformant plan*, i.e. a plan which contains provisions for any possibility and is guaranteed to reach the goal in any scenario. For situated

[1] This work has been partially supported by the EU-project SIARAS, Skill-Based Inspection and Assembly for Reconfigurable Automation Systems (FP6 - 017146).

agents, however, not only the task of *creating*, but even simply *storing*, such plan could exceed available resources.

Therefore, situated agents need to consciously alternate between reasoning, acting and observing their environment, or even do all those things in parallel. We aim to achieve this by making the agents create short partial plans and execute them, learning more about their surroundings throughout the process. They create several partial plans and reason about usefulness of each one, including what knowledge can it provide. They generalise their past experience to evaluate the likelihood of plans leading to the goal. The plans are conditional (i.e. actions to be taken depend on observations made during execution), which makes them more generic and means that their quality can be estimated more meaningfully. We also intend for the agent to judge by itself whether it is more beneficial to begin executing one of those plans immediately or rather to continue deliberation.

We expect the agent to live significantly longer than the duration of any single planning episode, so it should generalise solutions it finds. In particular, the agent needs to extract domain-dependent control knowledge and use it when solving subsequent, similar problem instances. It is the authors' belief that deductive knowledge, at least in many of the domains we are interested in, may contain more details and be more accurate than other forms of representation (such as numerical or probabilistic), therefore our agent learns deductively using a symbolic representation in Active Logic. To this end we introduce an architecture consisting of three modules, which allow us to combine state-of-the-art solutions from several fields of Artificial Intelligence, in order to provide the synergy our agent requires to achieve the desired functionality.

The goal of this paper is to show the results of experiments of using Inductive Logic Programming algorithm to evaluate partial plans within our architecture for situated rational agents. In the next section we introduce example domains on which we present our ideas. In section *Architecture* we describe the organisation of our agent. The three following sections introduce each of agent's functional modules in more detail: *Deductor*, *Actor* and *Learner*. After that, we presents the *Results* of our first experiments with the architecture, discuss some of the *Related Work* and finish with some *Conclusions*.

2 Experimental Domains

Throughout this paper we will be using a simple game called Wumpus, the well-known testbed for intelligent agents [2], 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 — remain a serious challenge.

The game is played on a square board. There are two characters, the player and the Wumpus. The player can, in each turn, move to any neighbouring square, while the Wumpus does not move at all. Position of the monster is not known to the player, he only knows that it hides somewhere on the board. Luckily, Wumpus is a smelly beast, so whenever the player enters some 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, by moving throughout the board and observing on which squares does it smell. At the

same time, if the player enters the square occupied by the beast, he gets eaten and loses the game.

For learning experiments we also use a second domain, a modified version of "king and rook vs king and knight" chess ending. Since we are interested in partially unknown environments we assume, for the sake of experimentations, that the agent does not know how the opponent's king is allowed to move — *a priori*, any move is legal. The agent will need to use learning to discover what kinds of moves are actually possible.

In order to understand the goal of our research, it can be helpful to imagine the setting somewhat akin to the *General Game Playing Competition* [3]: our agent is given some knowledge about the domain and is supposed to act rationally from the very beginning, while becoming more and more proficient as it gathers more experience.

3 Agent Architecture

The architecture of our agent consists of four main functional modules. Each of them is responsible for a different part of agent's rationality, but the overall intelligence is only achievable by the interactions of them all.

The *Deductor* module is the one responsible for classical "reasoning". It uses a logical formalism based on combination of Active Logic and Situation Calculus (as introduced in [4]) in order to find out consequences of the agent's current beliefs. Based on the domain knowledge and previous observations, it analyses possible actions and predicts what will be the effect of their execution.

The second module is a *Planner*, which generates partial, conditional plans applicable in the agent's current situation. In the reported experiments, our planner is a simple one, which generates pre-arranged plans only (all imaginable plans for the Wumpus domain, and some set of "interesting" plans for Chess). The third main module, *Actor*, oversees Deductor's reasoning process and evaluates plans the latter has come up with, trying to find out which is the most useful one to perform. For this paper, Actor waits until Deductor terminates and only executes plans after this happens, but in general it is Actor's responsibility to balance acting and deliberation.

Finally, the *Learner* module analyses the agent's past experience and induces rules for estimating quality of plans. Results of learning process are used both by Deductor and by Actor. In particular, since the plans Deductor reasons about are partial (i.e. they do not — most of the time — lead all the way to the goal) it can be very difficult to estimate whether a particular plan is a step in the right direction or not. Using machine learning techniques is one way in which this could be achieved.

In general, 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, both ones done in the past and those still in progress, the vast majority of AI research is being done in specialised subfields and it is our 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.

4 Deductor

Deductor performs logical inference and directly reasons about the agent's knowledge. In particular, it is the module which analyses both current state of the world and how it will change as a result of performing a particular action. To this end, the agent uses a variant of Active Logic [5], augmented with some ideas from Situation Calculus [6].

Active Logic [5] is a reasoning formalism which, unlike classical logic, concerns the *process* of performing inferences, not just the final extension of the entailment relation. In particular, instead of classical notion of theoremhood, AL has i*-theorems*, i.e. formulae which can be proven *in* i *steps*. This allows an agent to reason about *difficulty* of proving something, to retract knowledge found inappropriate and to resolve contradictions in a meaningful way, as well as makes the agent aware of the passage time and its own non-omniscience.

Following ideas of [4] we have decided to augment Active Logic with some concepts from Situation Calculus. In particular, in order to have the agent reason about changing world, every formula is indexed with current situation. Furthermore, since the agent needs to reason about effects of executing various plans, we additionally index formulae with the plan the agent is considering. Therefore, a typical formula our agent reasons about looks like this: $Knows(s, p, Neighbour(a2, b2))$, meaning "an agent knows that after executing plan p in situation s, squares $a2$ and $b2$ will be adjacent."

From agent's point of view, the most interesting formulae are ones of the form: $Knows(s, p, Wumpus(b3)) \vee Knows(s, p, Wumpus(c2))$, meaning "an agent knows that after executing plan p in situation s, it will *either* know that there is Wumpus on $b3$ or that there is Wumpus on $c2$". Which of the "or" clauses will be true depends on the observations that agent will make while acting. This is exactly the kind of knowledge that agent is interested in — it *does* tell important things about quality of the plan being considered. For a human "expert," such p looks like a good plan. The goal of our research is to make *an agent* be able to reason about plans in exactly this way.

In the experiments reported in this paper, we consider plans of length one and two only. In order to make plan evaluation more meaningful, we allow those plans not only to be simple (sequential) but also *conditional*, i.e. to have branches where actions depend on agent's observations. We expect that such conditional plans will be, in many domains, much easier to classify as either good or bad ones.

5 Planner and Actor

The Actor module is an overseer of Deductor and works as a controller of the agent as a whole. In its ultimate form, it is expected to do three main things. First, it guides the reasoning process by making it focus on the plans most likely to be useful. Second, it decides when enough time has been spent on deliberation and no further interesting results are likely to be obtained. Third, it makes decisions to execute a particular plan from Deductor's repertoire.

In this paper we have decided to focus on the interactions between learning and deduction, so both Planner and Actor have been significantly simplified. Planner does not use any heuristics and simply creates all possible plans, although we aim to use

existing planners to efficiently create only the "reasonable" plans. Deductor uses an incomplete reasoner which always terminates, therefore Actor does not decide *when* to begin plan execution — it simply lets Deductor infer everything it can about each of the available plans and chooses the best one based on all the available information.

6 Learner

The ultimate goal of the learning module is to provide Actor with knowledge necessary to choose the best plan for execution and to stop deliberation when too much time has been spent on it without any new interesting insights.

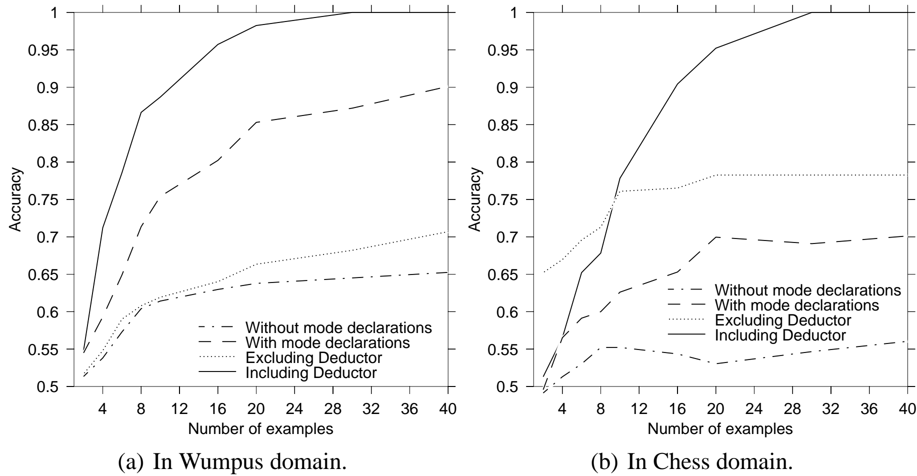
A step in this direction is to learn how to detect "bad" plans early, so that Deductor does not waste time deliberating about them. In our experimental domains we have defined bad plans to be those which can kill the agent (for Wumpus), and those that lead to losing the rook (for Chess).

In the next section 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. 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.

One question is how to represent situations and plans in the way most suitable for learning. We have decided to encode plan and its branches as additional arguments to the domain predicates. 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 consists of two branches (called *left* and *right*, with *left* being taken iff it smells on the newly-visited square). For Chess, there are three explicit branches (each specifying 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 representation is sufficiently general to work well across many different domains. For example, a predicate $Position(p1, left, a2)$ means that whenever the *left* branch of plan *p1* is executed, the agent will be on square *a2*.

In the experiments reported here, we assume that the agent has perfect knowledge about which plans (training examples) are bad ones. This is a justified assumption for 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. In Wumpus, the distinction is not so clear — it is possible that the agent will get lucky and not die even though it executes a dangerous plan, simply because the beast is in a favourable position.

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 about 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.



(a) In Wumpus domain. (b) In Chess domain.

7 Results of the experiments

For our experiments, we have used the Inductive Logic Programming algorithm called PROGOL [7], since it is among the best known ones and its author has provided a fully-functional, publicly available implementation. PROGOL is based on the idea of *inverse entailment* and it employs a covering approach similar to the one used by FOIL [8], in order to generate hypothesis consisting of a set of clauses which cover all positive examples and do not cover any negative ones.

We have used three example runs of a Wumpus game on a very small, 3x3, board. The player had considered 134 plans in each case, which is the total number of length 2 plans (both simple and conditional ones) in four situations: the player started on $a1$, first moved to $a2$, then to $b2$, and finally to $c3$.

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 is on $c2$. The second and third runs were similar, except that Wumpus was on $a3$ and $c1$, respectively. In the Chess domain, we have used three board positions in which the white have a winning strategy and have hand-crafted 69 plans covering different possible types of situations, since it is obviously not feasible to analyse *all* possible plans in this domain.

In the first experiment, we used as little domain-specific knowledge as possible, in particular we have not provided any *mode declarations* for PROGOL. The goal of mode declarations is to reduce the hypothesis search space by limiting types of predicate arguments, as well as by specifying which ones are input and which are output arguments and whether variables or constants should be used.

We have plotted the accuracy of the learned hypothesis as the lowest curve (marked “Without mode declarations”) for Wumpus in figure 7a and for Chess in figure 7b. Each point on the graph is an average over 50 trials. It can be easily seen that learning quality is too low to be practically useful.

The second curve (marked “With mode declarations”) clearly shows that providing even such a small amount of domain knowledge (mode declarations are easy to specify for a domain expert) is enough to greatly improve quality of learned hypothesis. It can

badPlan(A) $\Leftarrow$	visit(A,first,B), maybeWumpus(A,B).
badPlan(A) $\Leftarrow$	visit(A,second,B), maybeWumpus(A,B).
badPlan(A) $\Leftarrow$	visit(A,left,B), maybeWumpus(A,B).
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 1. Correct definitions of bad plans for Wumpus and Chess.

be also easily seen that the accuracy in the Wumpus domain is significantly higher than the one in the Chess domain. Nevertheless, the learning is still not fully successful — even though all the knowledge necessary to express the correct hypothesis is available. We attribute this to two things: overfitting and the fact that the search space is too large for PROGOL to handle sufficiently well.

Because of that, we have looked into ways of limiting the amount of knowledge used for learning — seemingly, presenting all of the agent's knowledge to the ILP algorithm is not the best idea. As a start, we have decided to use only the initial domain definition and the observations that the agent made in previous situations. In the Wumpus domain this resulted in knowledge base containing mostly $maybeSmells$, $knowsClear$ and $knowsSmell$ predicates, while in Chess mostly $position$, $canMove$ and geometrical relations. The results of learning can be seen on curve marked "Excluding Deductor", so named since they roughly correspond to an agent who does not have a specialised deduction module and uses learning only.

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 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. 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 develop for integration of different Artificial Intelligence areas is a useful one.

Finally, in our fourth and final 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 $maybeWumpus$, $noWumpus$ and $knowsWumpus$ predicates, while in Chess it included $notProtected$, $distanceTwo$, and $distanceTwo$ predicates. As can be seen from the curve marked "Including Deductor", the agent managed to perfectly identify bad plans from as few as 30 examples chosen at random, in both domains. The exact hypotheses that PROGOL learned are presented in table 1.

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* learning examples in an intelligent way.

Having established that successful learning is possible, one more thing that should be shown is whether it is actually *useful*. In our implementation (which is designed for flexibility of reasoning rather than for its speed) analysing a complete game of Wumpus takes (depending on the monster's real position) on the order of 15 hours. If Actor

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 2. Usefulness of learning.

knows how to identify bad plans and forces Deductor to ignore them, the total time drops down dramatically, to about *six hours*. This is a clear confirmation of our claim that the knowledge gained due to learning from experience can be very useful in improving efficiency of reasoning.

Finally, we would like to point out that PROGOL algorithm, while a very efficient one, is rather poorly suited for the class of problems we face. It was sufficient for a proof of concept and to show the general usefulness of learning as such, but our next step will be to find (or, more likely, develop) a different algorithm, better adapted to the particular needs of evaluating plans.

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.

The first to mention is [9], which 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.

There has been significant amount of work done in learning about what actions to take in a particular situation. One notable example is [10], where author showed important theoretical results about PAC-learnability of action strategies in various models. In [11] 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, [12] 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 work close to this approach is documented in [13], where *teleoreactive logic programs*, possibly even recursive ones, are used for representing the action part of an agent. On top of it a learning mechanism, quite similar to ILP, is employed for improving the existing action programs.

One attempt to escape the trap of large search space has been presented in [14], 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 [15].

The work mentioned above focuses primarily on learning how to act, without trying to reach conclusions in a deductive way. In a sense, the results are more similar to the reactive-like behaviour than to classical planning system, with many similarities to the reinforcement learning.

Outside the domain of planning, there is a lot of important research being done in the learning paradigm. Recently, [16] 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 [17], where the system learns domain-dependent control knowledge beneficial in planning tasks. From another point of view, [18,19] presented 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.

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 are the terms used by various researchers [20,21,22]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 [23] or earlier in [24]. We are not aware of any such research that would attempt to integrate learning into this approach.

9 Conclusions

We are developing an architecture for rational agents that combine planning, deductive reasoning, inductive learning and time-awareness in order to operate successfully in dynamic environments. Our agent creates conditional partial plans, reasons about their consequences using an extension of Active Logic with Situation Calculus features, and employs ILP learning to generalise past experience in order to distinguish good plans from bad ones.

In this paper we report on our experiments with using PROGOL learning algorithm to identify bad plans early, in order to save agent the (wasteful) effort of deliberating about them. We analyse how the quality of learning depends on the amount of additional domain-specific knowledge provided by the user specifically for the purpose of experience generalisation. Finally, we show that successful learning can result in a dramatic decrease of the agent's reasoning time.

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 data presented to it. In particular, the fact that different parts of knowledge base can have different "usefulness" or "relevance" is very important and should be taken into account explicitly — and, if possible, automated in some way.

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

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

Finally, the architecture we are presenting here is still evolving and the functionality of every module will be expanded in the future.

References

1. Chong, W., O'Donovan-Anderson, M., Okamoto, Y., Perlis, D.: Seven days in the life of a robotic agent. In: GSFC/JPL Workshop on Radical Agent Concepts. (2002)
2. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edn. Prentice Hall Series in AI (2003)
3. Genesereth, M., Love, N., Pell, B.: General game playing: Overview of the aaai competition. AI Magazine **26**(2) (2005) 62–72
4. Nowaczyk, S.: Partial planning for situated agents based on active logic. In: Workshop on Logics for Resource Bounded Agents, ESSLLI 2006. (2006)
5. Purang, K., Purushothaman, D., Traum, D., Andersen, C., Perlis, D.: Practical reasoning and plan execution with active logic. In Bell, J., ed.: Proceedings of the IJCAI-99 Workshop on Practical Reasoning and Rationality. (1999) 30–38
6. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press (2001)
7. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming **13**(3-4) (1995) 245–286
8. Mitchell, T.M.: Machine Learning. McGraw-Hill Higher Education (1997)
9. Dietterich, T.G., Flann, N.S.: Explanation-based learning and reinforcement learning: A unified view. In: International Conference on Machine Learning. (1995) 176–184
10. Khardon, R.: Learning to take actions. Machine Learning **35**(1) (1999) 57–90
11. Moyle, S.: Using theory completion to learn a robot navigation control program. In: ILP. (2002)
12. Könik, T., Laird, J.E.: Learning goal hierarchies from structured observations and expert annotations. Machine Learning **64** (2006) 263–287
13. Langley, P., Choi, D.: Learning recursive control programs from problem solving. Journal of Machine Learning Research **7** (2006) 493–518
14. Džeroski, S., Raedt, L.D., Driessens, K.: Relational reinforcement learning. Machine Learning **43**(1/2) (2001) 7–52
15. Morales, E.F.: Relational state abstractions for reinforcement learning. In: ICML-04 Workshop on Relational Reinforcement Learning. (2004)
16. Colton, S., Muggleton, S.: ILP for mathematical discovery. In: 13th International Conference on Inductive Logic Programming. (2003)
17. Fern, A., Yoon, S., Givan, R.: Learning domain-specific control knowledge from random walks. In: International Conference on Automated Planning and Scheduling. (2004)
18. Khardon, R., Roth, D.: Learning to reason with a restricted view. In: Workshop on Computational Learning Theory. (1995)
19. Khardon, R., Roth, D.: Learning to reason. Journal of the ACM **44**(5) (1997) 697–725
20. Cimatti, A., Roveri, M., Bertoli, P.: Conformant planning via symbolic model checking and heuristic search. Artificial Intelligence **159**(1-2) (2004) 127–206
21. Petrick, R.P.A., Bacchus, F.: Extending the knowledge-based approach to planning with incomplete information and sensing. In: International Conference on Automated Planning and Scheduling. (2004) 2–11
22. van der Hoek, W., Wooldridge, M.: Tractable multiagent planning for epistemic goals. In: Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2002)

23. Bonet, B., Geffner, H.: Planning and control in artificial intelligence: A unifying perspective. Applied Intelligence **14**(3) (2001) 237–252
24. Dean, T., Wellman, M.P.: Planning and Control. Morgan Kaufmann (1991)