# An Architecture for Resource Bounded Agents

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 agents employ 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 agents can choose the best plan for execution.
We describe an architecture which allows ideas and solutions from several subfields of Artificial Intelligence to be joined together in a controlled and manageable way. In our opinion, no situated agent can achieve true rationality without using at least logical reasoning and learning. In practice, it is clear that pure logic is not able to cope with all the requirements put on reasoning, thus more domain-specific solutions, like planners, are also necessary. Finally, any realistic agent needs a reactive module to meet demands of dynamic environments.
Our architecture is designed in such a way that those three elements interact in order to complement each other's weaknesses and reinforce each other's strengths.[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 domains still remain a major 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, hand-crafted 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]. To facilitate this, we use Active Logic [2] for knowledge representation, which characterises reasoning as an ongoing process, instead of focusing on a fixed point of entailment relation.

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

Due to limited resources and to the necessity to stay responsive in a dynamic world, situated agents cannot be expected to create a complete plan for achieving their goals. In theoretical AI a common approach is to create a *conformant plan*, i.e. a plan which contains provisions for any possible sequence of external events and observations, and which is guaranteed to reach the goal in all scenarios. For situated agents, however, not only the task of *creating*, but even a requirement to simply *store* such plan could easily exceed available resources.

The lack of resources is partly addressed by considering "information-providing" actions and interleaving their execution with planning activity. In particular, executing them at the right time allows the agent to greatly simplify its subsequent planning process — it no longer needs to take into account the vast number of possible situations which would 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 tasks.

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 any particular plan 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 and, possibly, develop longer and more complete plans, in order to avoid making an unrecoverable mistake.

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 four 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 needs to achieve the desired functionality.

Ultimately, the agent will need to be able to handle non-stationary, adversary environment, to cooperate with others in multi-agent setting and to plan for goals more complex than simple reachability properties, such as temporally extended goals and restoration goals. It is our belief that symbolic knowledge representations are the only way of achieving such versatility.

The goal of this paper is to present and justify the architecture we use for our agents, as well as suggest possible ways to extend it. In the next section we introduce example domains which we exploit to present and test our ideas. In section *Architecture* we provide an overview of the organisation of our agent. The following four sections introduce each of agent's functional modules in more detail: *Deductor*, *Planner*, *Actor* and *Learner*. After that, we describe the module interaction in some more detail, briefly present some of the *Results* of our 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 two simple domains, to better illustrate our ideas. The first one is a simple game called Wumpus [3]. The game is 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 monster called 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 may be legal. The agent will need to use learning and discover what kinds of moves are actually possible.

Those domains are, obviously, only examples and the architecture presented here does not dependent on them. In order to better understand the goal of our research, it can be helpful to imagine the setting similar to the *General Game Playing Competition* [4]: our agent is given some declarative 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 Architecture

The architecture we propose consists of four main modules (shown in Figure 1). *Deductor* corresponds to a typical "core" of logically reasoning agent, except that in our case it does not use classical logic, but rather Active Logic formalism in order to better interact with other modules. *Planner* module is mainly responsible for creating potentially interesting plans, although any kind of domain-specific reasoning could just as well be performed in it. Its main purpose is to increase the amount of agent's beliefs (including plans) that provide Deductor with knowledge to reason about.

*Actor* is an overseer of the interactions between an agent and its environment. It analyses and interprets agent's sensor data, in order to react whenever something interesting happens in the external world. It also watches over agent's reasoning process and

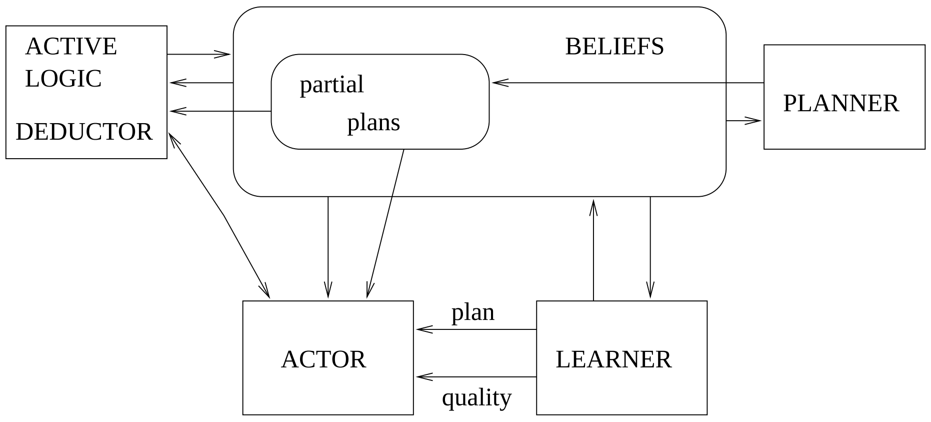


**Fig. 1.** Architecture of an agent.

makes decisions about when their results are sufficiently well developed to begin acting based on them.

Finally, *Learner* analyses the performance of the agent as a whole and generalises its past experience in order to improve the work of each module. In particular, the most important part is to evaluate conditional partial plans and to learn which ones are most likely to lead to good results in future situations.

In the following sections, we describe each module in more detail, as well as point out the most important interactions among them.

## 4 Deductor

This module analyses both current state of the world and how it will change as a result of performing a particular action. Our agent uses a variant of Active Logic [2], augmented with some ideas from Situation Calculus [5].

Active Logic is a reasoning formalism which, unlike classical logic, is concerned with the *process* of performing inferences, not just the final outcome (fixed point) 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, among other things, about *difficulty* of proving something, to retract knowledge found inappropriate and to resolve contradictions in a meaningful way, as well as makes it aware of the passage time and its own non-omniscience.

To this end, each formula in AL is labelled with the step number when it was derived. E.g., the *modus ponens* inference rule looks like this: $\frac{i:\alpha,\alpha\Rightarrow\beta}{i+1:\beta}$ and means "if at step $i$ formulae $\alpha$ and $\alpha \Rightarrow \beta$ are present in the belief set, then at step $i+1$ formula $\beta$ 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. A more in-depth description of Active Logic can be found in [6].

Following ideas of [7] 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: $Knows(s, p, Neighbour(a2, b2))$ and means "an agent knows that after executing plan $p$ in situation $s$, squares $a2$ and $b2$ will be adjacent[2]." This formula is only mildly interesting, as its validity depends on neither $s$ nor $p$, at least in the Wumpus domain. But: $Knows(s, p, \neg Wumpus(b2))$ which means "an agent knows that after executing plan $p$ in situation $s$, Wumpus will not be on $b2$," *does*, obviously, depend on $s$, since agent's knowledge changes as it acts in the world. It still does not, however, depend on $p$ itself. Clearly, no "new" knowledge can be obtained by simply *considering* some plan (without actually executing it). If an agent $Knows(s, p, \neg Wumpus(b2))$, then it must also $Know(s, \emptyset, \neg Wumpus(b2))$, where $\emptyset$ stands for an empty plan[3].

In contrast, an example of some really interesting formulae that can be formulated could be $Knows(s, p, Wumpus(b3)) \vee Knows(s, p, Wumpus(c2))$ which means "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 $c2$". As an example of reasoning by cases and predicting action results, this is exactly the kind of knowledge that we want the agent to infer — it *does* tell important things about quality of the plan being considered. If all the agent knew before was: $Knows(s, \emptyset, Wumpus(b3) \vee Wumpus(c2))$ than clearly executing $p$ is useful. 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. It is our intuition, supported by preliminary experiments reported later in this paper, that creating a *domain independent* Actor module which would efficiently select good plans by learning from experience using formulae like the one above is possible.

## 5 Planner

As we stated earlier, our agent creates conditional, partial plans. The plans are partial because limited resources do not allow our agent to consider all the possibilities and come up with a good conformant plan. The plans are conditional since 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 strictly more expressive than in the unconditional case. In particular, their applicability is significantly enhanced and the agent is able to reason about their usefulness.

Let us take an example from the Wumpus domain. With an agent on square $a1$, one simple plan is "$a2$", meaning "go to $a2$", while another is "$a2a3$", meaning "go to $a2$ and then go to $a3$". A conditional plan could be "$a2\ ?\ a1 : b2$", meaning "go to $a2$ and if it smells there go back to $a1$, else go forward to $b2$". Actually, in Wumpus domain it

[2] We use chess-like notation for naming squares, with letters designing columns and numbers designing rows.

[3] Since in our game the player has no way of changing Wumpus' position, the actual validity of "$Wumpus(b2)$" remains constant, only agent's knowledge is changing

is difficult to find a simple plan longer than one step which would be good, while there exists a number of conditional plans which can be easily classified as good.

In a sense, our treatment of plans is related to the notion of hierarchical planning, since the conditional partial plans we consider are very similar to macro-operators [8]. 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.

## 6 Actor

The Actor module acts as a controller of the agent as a whole. It has three kinds of duties. First, it observes the environment and forces the agent to react to interesting events that take place there. Second, it decides when enough time has been spent on deliberation and no further interesting results are likely to be obtained. Finally, it makes decisions to execute a particular plan from Deductor's repertoire.

The typical scenario consists of Actor continuously monitoring the sensor input, analysing it and transforming it into symbolic representation whenever needed. In Active Logic, there exists a special provision for that, called *observation function*, with exactly the same status as domain axioms, but with different temporal extent. Such input can then be used by the Deductor, allowing an agent to respond to the changes in the world. It is also possible for Actor to react more "violently" if needs arises, either by directly and immediately performing some physical actions (e.g. obstacle avoidance if the agent is about to hit something) or by straightforwardly influencing agent's internal reasoning process (for example, if Actor notices that the change of the environment reaches some threshold, it can "reset" Deductor, under the assumption that its previous results are no longer applicable anyway). In this sense, the Actor implements the reactive behaviour of the agent.

In addition, Actor continuously monitors the progress of agent's internal reasoning and can react to certain conditions there. For example, as soon as Deductor finds a plan which looks sufficiently good (in the extreme case, one which directly leads to the goal), the reasoning can be interrupted and this plan executed immediately. Similarly, if it turns out that the deduction cannot proceed further before some observation is made, Actor can decide to pause the reasoning and perform the necessary sensing actions.

Finally, Actor chooses which plan to execute. This decision may be based on plan ranking done earlier by either Deductor or Learner, or may simply be a random pick, if a situation is sufficiently new and such knowledge is not yet available.

## 7 Learner

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

In order to do that, Learner finds out recipes for evaluating plans and to decide which ones are most likely to lead to the goal. Since the plans are partial, it is very difficult to predict, in general, whether a given plan is a step in the right direction. There is an active research on heuristics in planning, focusing on exactly this problem.

Using learning techniques developed in the field of Inductive Logic Programming, however, is another possible way to approach the problem. Based on experience and on deductive reasoning performed by Deductor module, it is possible to 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 use that to choose the one to be executed next.

In the simplest case, it could consist of just distinguishing "dangerous" plans, i.e. ones that can lead to agent's immediate death. This is not enough in general, but there is a large class of domains which are "safe" in a sense that it is possible to win from every position. Wumpus domain belongs to this class, while chess does not.

Moreover, an important question is one of credit assignment, since the agent typically executes several partial plans before it reaches a 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, each with own advantages and disadvantages, being developed for dealing with this problem, and it is not clear at this point which one would be best suited for our particular case.

## 8 Module Interactions

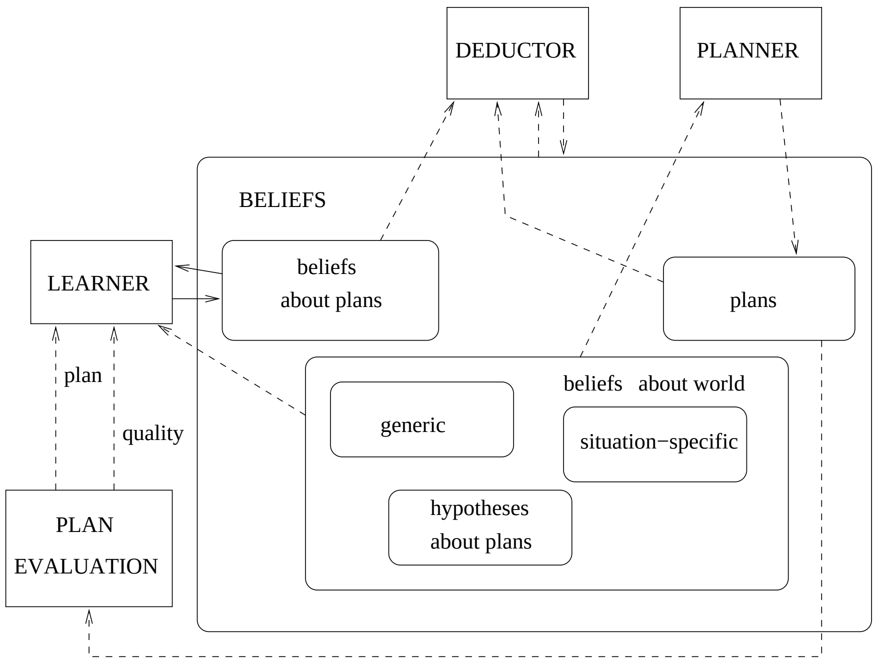


**Fig. 2.** Some details of the architecture of the agent.

The main idea of our architecture is to ensure fruitful interactions between the modules introduced above. Each of them roughly corresponds to one of the major subfields of Artificial Intelligence and is designed to employ state of the art solutions from it. Each module can provide a very good performance in specific situations, but neither is quite sufficient to achieve real intelligent behaviour of a generic situated agent.

The major contribution of our architecture is the idea to use multiple conditional partial plans, together with expected results of their application to a particular situation, as the way to exchange information between modules. In this way, Planner comes up with plans which 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 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. Actor predicts, based on that, which should be executed in the future, as well as oversees the reasoning and chooses the best course of action as soon as enough knowledge becomes available.

Using the above architecture, agent is able to reason about current state of the world, both about the details of current situation and about the generic laws governing the application domain. Moreover, it can also reason about the state of the world as it expects it to be after execution of each plan under consideration. Finally, it reasons bout *plans* themselves, how successful were they in the past, both in general and in situations similar to the current one, and how good they are expected to be right now.

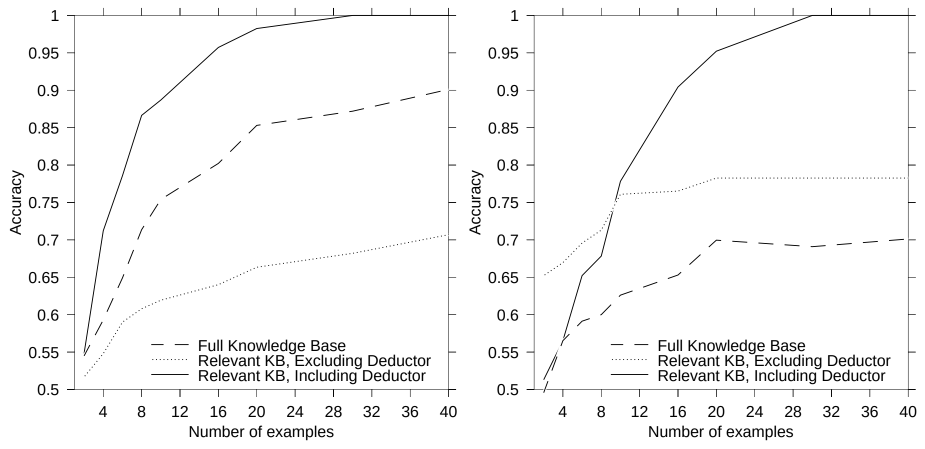
## 9 Results of Initial Experiments

We have performed some initial experiments in order to evaluate the feasibility of our ideas and to check how well do they work in practice. Our focus was on interactions between modules and on showing that different approaches we combine do indeed complement each other.

We have used an ILP algorithm PROGOL [9] for learning, since it is among the best known ones and its author has provided a fully-functional, publicly available implementation. In a previous paper [10] we have presented results of learning to distinguish “bad” plans early. We have shown 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 are known to be useless. Those results required providing additional domain knowledge specifically for the purpose of learning, but we have also analysed (in [11]) how the PROGOL learning algorithm can be adapted to extract it automatically.

The results we have obtained can be seen in figure 3. The first curve (middle one, marked “Full Knowledge Base”) corresponds to the case where we used no additional domain-specific knowledge, except from PROGOL *mode declarations*, and where the whole knowledge the agent has gathered has been provided to the learning algorithm.

For the remaining two curves we identified the parts of agent’s knowledge which are most relevant to learning the concept of bad plans and only presented those to the learning algorithm. In the lowest one, marked “Relevant KB, Excluding Deductor”, we have provided only domain description and observations, having removed all the results



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

**Fig. 3.** Results of learning.

of Deductor's work. In the highest one, we have left them intact. Only in this latest case the Learner has been able to correctly learn the concept.

Those results show that Deductor provides knowledge which significantly improves 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. We plan further experiments which shall reveal even more synergy between modules in the architecture.

## 10 Related Work

The amount of work on agent architectures in general is too large to summarise comprehensively here, therefore we will only discuss a small fraction of this field here.

The work presented in this paper is related to studies of architectures for general intelligence. There has been a lot of attempts to define such an architecture, with SOAR being the most prominent and most successful of them ([12] provides a general introduction). In contrast to SOAR, we do not claim any cognitive plausibility of our architecture, focusing our interests on mainly on the task of achieving intelligent behaviour of an artificial agent.

Our approach is also quite distinct from the layered architectures which are popular nowadays (see, for example, [13] for a collection of papers on this topic). In particular, we do not focus specifically on the reactive part of the system, hiding it one part of the Actor module. This does not mean that we diminish the necessity or importance of the 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 higher levels of our architecture remain sufficiently flexible to be able to handle the requirements of reactive part.

Although we have been stressing the need of rationality and the importance of reasoning throughout this paper, our approach also differs from the one exhibited by Beliefs-Desires-Intentions systems and BDI architectures (see, among others, [14] for a formalised approach to this topic). We do not explicitly distinguish intentions, while both plans and goals are treated in a very similar manner. This is due to the fact, however, that BDI approaches usually lack learning, which is the central issue in our approach, and which allows an agent to account for its intentions in a different, but also sufficiently effective, way.

## 11 Conclusions

In this paper we present an architecture for rational agents we are currently developing. It allows agents to combine planning, deductive reasoning, inductive learning and time-awareness in order to operate successfully in dynamic environments. Agents create conditional partial plans, reason about their consequences using an extension of Active Logic with Situation Calculus features, and employ ILP learning to generalise past experience in an attempt to distinguish good plans from bad ones.

The basic framework of the architecture is already implemented and initial experiments show that the synergy effects we aimed for do, indeed, appear. The modules themselves still need more work before they contain the functionality needed to tackle practical problems, since we have only been working on toy ones up to now. State of the art solutions, however, are available and many of them can be integrated into our architecture with minimal changes. Nevertheless, the ones needing further attention are efficient Deductor for Active Logic, and Learner using modified ILP algorithm. Our current research focuses on those topics.

## 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. Elgot-Drapkin, J., Kraus, S., Miller, M., Nirkhe, M., Perlis, D.: Active logics: A unified formal approach to episodic reasoning. Technical report, University of Maryland (1999)
3. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edn. Prentice Hall Series in AI (2003)
4. Genesereth, M., Love, N., Pell, B.: General game playing: Overview of the aaai competition. AI Magazine **26**(2) (2005) 62–72
5. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press (2001)
6. 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
7. Nowaczyk, S.: Partial planning for situated agents based on active logic. In: Workshop on Logics for Resource Bounded Agents, ESSLLI 2006. (2006)
8. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
9. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming **13**(3-4) (1995) 245–286

10. Nowaczyk, S., Malec, J.: Learning to evaluate conditional partial plans. In: Sixth International Conference on Machine Learning and Applications. (2007)
11. Nowaczyk, S., Malec, J.: Inductive logic programming algorithm for estimating quality of partial plans. In: Mexican International Conference on Artificial Intelligence. (2007)
12. Lewis, R.L.: Cognitive theory, SOAR. International Encylopedia of the Social and Behavioral Sciences. Amsterdam (2001)
13. Kortenkamp, D., Bonasso, R.P., Murphy, R., eds.: Artificial Intelligence and Mobile Robots. AAAI Press / MIT Press (1998)
14. Wooldridge, M.: Reasoning about Rational Agents. MIT Press (2000)