Cognitive Robotics*

Hector J. Levesque
Dept. of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 3A6
hector@cs.toronto.edu

Gerhard Lakemeyer
Dept. of Computer Science
RWTH Aachen
52056 Aachen
Germany
gerhard@cs.rwth-aachen.de

This chapter is dedicated to the memory of Ray Reiter. It is also an overview of cognitive robotics, as we understand it to have been envisaged by him.[1] Of course, nobody can control the use of a term or the direction of research. We apologize in advance to those who feel that other approaches to cognitive robotics and related problems are inadequately represented here.

1 Introduction

In its most general form, we take *cognitive robotics* to be the study of the knowledge representation and reasoning problems faced by an autonomous robot (or agent) in a dynamic and incompletely known world. To quote from a manifesto by Levesque and Reiter [42]:

> "Central to this effort is to develop an understanding of the relationship between the knowledge, the perception, and the action of such a robot. The sorts of questions we want to be able to answer are
>
> - to execute a program, what information does a robot need to have at the outset vs. the information that it can acquire *en route* by perceptual means?
> - what does the robot need to know about its environment vs. what need only be known by the designer?
> - when should a robot use perception to find out if something is true as opposed to reasoning about what it knows was true in the past?
> - when should the inner workings of an action be available to the robot for reasoning and when should the action be considered primitive or atomic?

[0]Reprinted from: Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation,* Chapter 23, pp. 869–886, Copyright (2007), with permission from Elsevier.

[1]To the best of our knowledge, the term was first used publicly by Reiter at his lecture on receiving the IJCAI Award for Research Excellence in 1993.

Dagstuhl Seminar Proceedings 10081
Cognitive Robotics
http://drops.dagstuhl.de/opus/volltexte/2010/2633

and so on. With respect to robotics, our goal (like that of many in AI) is *high-level robotic control*: develop a system that is capable of generating actions in the world that are appropriate as a function of some current set of beliefs and desires. What we do *not* want to do is to simply engineer robot controllers that solve a class of problems or that work in a class of application domains. For example, if it turns out that online reasoning is unnecessary for some task, we would want to know what it is about the task that makes it so."

We take this idea of knowledge representation and reasoning *for the purpose of* high-level robotic control to be central to cognitive robotics [75]. This connects cognitive robotics not only to (traditional, less cognitive) robotics but also, as discussed later, to other areas of AI such as planning and agent-oriented programming.

To illustrate the knowledge representation and reasoning issues relevant to high-level robotic control, we will use Reiter's variant of the situation calculus. There are several reasons for this: we, the authors, have worked with the situation calculus and hence feel most comfortable with it; the situation calculus is a very expressive formalism which can be used to model many of the features relevant to cognitive robotics; it was already introduced at length in a chapter of this volume (which we assume as a prerequisite), so that we do not need to present it from scratch; and last but not least, it is a tribute to Ray Reiter. For a book length treatment of cognitive robotics *not* based on the situation calculus, see [85].

The structure of the this chapter is as follows. In Section 2, we discuss some of the knowledge representation issues that arise in the context of cognitive robotics. In Section 3, we turn to problems in automated reasoning in the same setting. In Section 4, we examine how knowledge representation and reasoning come to bear on the issue of high-level agent control. Finally, in Section 5, we briefly draw conclusions and suggest a direction for future research.

2 Knowledge representation for cognitive robots

As a special sort of knowledge-based system, cognitive robots need to represent knowledge about relevant parts of the world they inhabit. What makes them special is the emphasis on knowledge about the *dynamics* of the world, including, the robot's own actions. In currently implemented systems, knowledge about objects in the world can be very simple, as in robotic soccer [21], where little is known beyond their position on a soccer field, to the very complex, involving knowledge about the actual shape of the objects [60, 71]. Likewise, knowledge about actions can be as simple as taking an action to be a discrete change of position from A to B, or fairly involved with probabilistic models of success and failure [23, 22].

But whatever the application, the key feature of cognitive robotics is the focus on a changing world. A suitable knowledge representation language must at the very least provide *fluents*, that is, predicate or function symbols able to change their values as a result of changes in the world. For our purposes, we will use the situation calculus; but there are many other possible choices, modal vs. non-modal, state-based vs. history-based, time-based vs. action-based, and so on.[2] Each of these will need to address similar sorts of

[2]While planning languages like STRIPS [28] or PDDL [57] also qualify and have been used to control

issues such as the frame, qualification, and ramification problems, discussed in the Situation Calculus chapter, and in [70].

2.1 Varieties of actions

In its simplest setting, the situation calculus is used to model actions that change the world in a discrete fashion and instantaneously. For robotic applications, this is usually far too limited and we need much richer varieties. Let us begin with actions which are continuous and have a duration. A simple idea to accommodate both is due to Pinto [58], who proposed to split, say, a *pickup* action into two (instantaneous) *startPickup* and *endPickup* actions with an additional time argument and a new fluent *Pickingup* with the following successor state axiom:

$$\begin{aligned} \mathit{Pickingup}(x,t,do(a,s)) \;\equiv\; & \exists t'(a = \mathit{startPickup}(x,t') \wedge t' \leq t) \\ \vee \quad & \mathit{Pickingup}(x,t,s) \wedge \neg\exists t'(a = \mathit{endPickup}(x,t') \wedge t' \leq t). \end{aligned}$$

While this works fine for some applications,[3] having to explicitly specify time points when an action starts and ends is often cumbersome if not impossible. An alternative approach, first introduced by Pinto [58] and later adapted by Grosskreutz and Lakemeyer [30] is to define fluents as continuous functions of time. For example, a robot's location while moving may be approximated by a linear function taking as arguments the starting time of the moving action and the robot's velocity. Using the special action called *waitFor*(ϕ) time advances until the condition ϕ becomes true. The use of *waitFor* was actually inspired by robot programming languages like RPL [53]. For an approach to continuous change in the event calculus see [72].

The situation calculus also deals with actions whose effects are deterministic, that is, where there is no doubt as to which fluents change and which do not. In practice, however, the world is often not that clear cut. For example, the robot's gripper may be slippery and the *pickup* action may sometimes fail, that is, sometimes it holds the object in its gripper afterwards and sometimes it does not. There have been a number of proposals to model nondeterministic effects such as [82, 27, 4]. On a more fine-grained level, which is often more appropriate in robotics applications, one also attaches probabilities to the various outcomes. Reiter's stochastic situation calculus [66], for example, achieves this by appealing to nature choosing among various deterministic actions according to some probability distribution. For example, imagine that when the robot executes a *pickup* action, nature actually chooses one of two deterministic actions *pickupS* and *pickupF*,which stand for a successful and failed attempt and which occur, say, with probabilities .95 and .05, respectively. A nice feature of this approach is that successor state axioms can be defined as usual because they only appeal to nature's choices, which are then deterministic.

robots [55, 61, 20], they are more limited in that they only specify planning problems, but do not lend themselves to a general representation and reasoning framework for cognitive robots as advocated by Reiter.

[3] Thinking of *all* actions as instantaneous in this way also has the advantage of reducing the need for true action parallelism, allowing us to use the much simpler variant of interleaved concurrency [17].

2.2 Sensing

In the situation calculus, actions are typically thought of as changes to the world, in particular, those which are due to a robot's actuators. Sensing actions, which provide the robot with information about what the world is like but leave the world unchanged otherwise, are of equal importance from a robot's perspective. Various ways to model sensing in the situation calculus have been proposed. One is to introduce a special fluent $SF(a, s)$ (for *sensed fluent value*) and axioms describing how the truth value of *SF* becomes correlated with those aspects of a situation which are being sensed by action a [41]. For example, suppose we have a sensing action $senseRed(x)$, which registers whether the colour of object x is red. This can be captured by the following axiom:

$$SF(senseRed(x), s) \equiv Colour(x, red, s).$$

The idea is that, when the robot executes *senseRed*, its sensors or perhaps more concretely, its image processing system, returns a truth value, which then tells the robot whether the object in question is red. We can use this predicate to define what the robot learns by doing actions $a_1, a_2, \ldots, a_n$ in situation s and obtaining binary sensing results $r_1, r_2, \ldots, r_n$:

$$\begin{aligned}
&Sensed(\langle\,\rangle, \langle\,\rangle, s) \stackrel{\text{def}}{=} True;\\
&Sensed(\vec{a} \cdot A, \vec{r} \cdot 1, s) \stackrel{\text{def}}{=} SF(A, do(\vec{a}, s)) \wedge Sensed(\vec{a}, \vec{r}, s);\\
&Sensed(\vec{a} \cdot A, \vec{r} \cdot 0, s) \stackrel{\text{def}}{=} \neg SF(A, do(\vec{a}, s)) \wedge Sensed(\vec{a}, \vec{r}, s).
\end{aligned}$$

In general, of course, sensing results are not binary. For example, reading the temperature could mean returning an integer or real number. See [79] on how these can be represented. Noisy sensors can be dealt with as well, as shown in [3, 73]. For the distinction between sensing and perception, see [59].

Sensing the colour of an object is usually deliberate, that is, the robot chooses to actively execute an appropriate sensing action. There are, however, cases where sensing results are provided in a more passive fashion. Consider, for example, a robot's need to localize itself in its environment. In practice, this is often achieved using probabilistic techniques such as [86], which continuously output estimates of a robot's pose relative to a map of the environment. Grosskreutz and Lakemeyer [32] show how to deal with this issue using so-called *exogenous* actions. These behave like ordinary non-sensing actions, which change the value of fluents like the robot's location. The only difference is that they are not issued by the robot "at will," but are provided by some external means. See also [15, 68] for how passive sensors can be represented by other means. Exogenous actions are not limited to account for passive sensing. In general, they can be used to model actions which are not under the control of the robot, including those performed by other agents.

2.3 Knowledge

When a robot has a model of its environment in the form of, say, a basic action theory, this represents what the agent *knows* or *believes* about the world. Yet so far there is no explicit notion of knowledge as part of the theory, and this may not be necessary, if we are interested only in the logical consequences of that theory. However, this changes when we need to refer to what the robot does *not* know, which is useful, for example, when

deciding whether or not to sense. We need an explicit account of knowledge also when it comes to knowledge about the mental life (including knowledge) of other agents. In the situation calculus, knowledge is modeled possible-world style[4] by introducing a special fluent $K(s', s)$, which is read as "situation s' is (epistemically) accessible from s." Let $\phi[s]$ be a formula that is uniform in s. Then knowing ϕ at a situation s, written as $\mathit{Knows}(\phi, s)$, means that ϕ is true in all accessible situations:

$$\mathit{Knows}(\phi, s) \stackrel{\text{def}}{=} \forall s'.K(s', s) \supset \phi[s'].$$

This idea of reifying possible worlds was first introduced by Moore [54]. Later, Scherl and Levesque [79] showed that the way an agent's knowledge changes as a result of actions can be captured by a successor state axiom for the fluent K:

$$K(s'', \mathit{do}(a, s)) \equiv \exists s'.s'' = \mathit{do}(a, s') \wedge K(s', s) \wedge [\mathit{SF}(a, s') \equiv \mathit{SF}(a, s)].$$

In words: a situation s'' is accessible after action a is performed in s just in case it is the result of doing a in some other situation s' which is accessible from s and which agrees with s on the value of *SF*. The effect of this axiom is, roughly, that it eliminates from further consideration all those situations which disagree with the result of sensing. For example, if a $\mathit{senseRed}(A)$ action returns the value *true*, only those situations remain accessible after performing the action where A is red. Note that this notion of epistemic alternatives generalizes the situation calculus discussed in the chapter of this volume in that we now assume that there are initial situations other than S_0.[5] One nice feature of the successor state axiom for K is that general properties of the accessibility relationship like reflexivity or transitivity only need to be stipulated for initial situations, as they are guaranteed to hold ever after [79]. For a treatment of knowledge and sensing in the fluent calculus, see [83]. For approach to knowledge in the situation calculus that avoids using additional situations, see [19].

Besides knowledge, there are many other mental attitudes that a cognitive robot may find useful to model. Proposals exist, for example, to model *goal* or *ability*, also using a possible-world semantics [78, 39, 50, 36]. The issue of belief change after receiving information that conflicts with what is currently known about the world has also been addressed [76, 77]. Here a preference relation over situations plays an essential role.

3 Reasoning for cognitive robots

The research problems in cognitive robotics are not limited to problems in representation seen in the previous section. We are fundamentally concerned with how these representations are to be *reasoned* with, and furthermore, as we will see in the next section, how this reasoning can be used to control the behaviour of the robots.

3.1 Projection via progression and regression

There are two related reasoning tasks that play a special role in cognitive robotics. The main one is called the (temporal) *projection task*: determining whether or not some condition

[4] Modeling knowledge using possible worlds is due to Hintikka [35].

[5] Instead of a single tree rooted at S_0, we now have a forest of trees each with their own initial situation.

will hold after a sequence of actions has been performed starting in some initial state. The second one is called the *legality task*: determining whether a sequence of actions *can* be performed starting in some initial state. Assuming we have access to the preconditions of actions, legality reduces to projection, since we can determine legality by verifying that the preconditions of each action in the sequence are satisfied in the state just before the action is executed. Projection is a very basic task since it is necessary for a number of other larger tasks, including planning and high-level program execution, as we will see in the next section.

We can summarize the definition of projection from the Situation Calculus chapter as follows: given an action theory $\mathcal{D}$, a sequence of ground action terms, $a_1, \ldots, a_n$, and a formula $\phi[s]$ that is uniform in s, the task is to determine whether or not

$$\mathcal{D} \models \phi[do(\vec{a}, S_0)].$$

As explained in that chapter, one of the main results proved by Reiter in his initial paper on the frame problem [65] is that the projection problem can be solved by *regression*: when $\mathcal{D}$ is a basic action theory (as defined in the earlier chapter), there is a regression operator $\mathcal{R}$, such that for any ϕ uniform in s,

$$\mathcal{D} \models \phi[do(\vec{a}, S_0)] \quad \text{iff} \quad \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \phi'[S_0],$$

where $\mathcal{D}_{S_0}$ is the part of $\mathcal{D}$ that characterizes S_0, and $\phi' = \mathcal{R}(\phi, \vec{a})$. So to solve the projection problem, it is sufficient, to regress the formula using the given actions, and then to determine whether result holds in the initial situation, a much simpler entailment.

Regression has proven to be a powerful method for reasoning about a dynamic world, reducing it to reasoning about a static initial situation. However, it does have a serious drawback. Imagine a long-lived robot that has performed thousands or even millions of actions in its lifetime, and which at some point, needs to determine whether some condition currently holds. Regression involves transforming this condition back through the thousands or millions of actions, and then determining whether the transformed condition held initially. This is not an ideal way of staying up to date.

The alternative to regression is *progression*. In this case, we look for a progression operator $\mathcal{P}$ that can transform an initial database $\mathcal{D}_{S_0}$ into the database that results after performing an action. More precisely, we want to have that

$$\mathcal{D} \models \phi[do(\vec{a}, S_0)] \quad \text{iff} \quad \mathcal{D}_{una} \cup \mathcal{D}'_0 \models \phi[S_0],$$

where $\mathcal{D}_{S_0}$ is the part of $\mathcal{D}$ that characterizes S_0, and $\mathcal{D}'_0 = \mathcal{P}(\mathcal{D}_{S_0}, \vec{a})$. The idea is that as actions are performed, a robot would change its database about the initial situation, so that to determine if ϕ held after doing actions $\vec{a}$, it would be sufficient to determine if ϕ held in the progressed situation (with no further actions), again a much simpler entailment. Moreover, unlike the case with regression, a robot can use its *mental idle time* (for example, while it is performing physical actions) to keep its database up to date. If it is unable to keep up, it is easy to imagine using regression until the database is fully progressed.

There are, however, drawbacks with progression as well. For one thing, it is geared to answering questions about the *current* situation only. In progressing a database forward, we effectively lose the historical information about what held in the past. It is, in other words,

a form of *forgetting* [48, 38]. While questions about a current situation can reasonably expected to be the most common, they are not the only meaningful ones.

A more serious concern with progression is that it is not always possible. As Lin and Reiter show [49], there are simple cases of basic action theories where there is no operator $\mathcal{P}$ with the properties we want. (More precisely, the desired $\mathcal{D}_0'$ would not be first-order definable.) To have a well-defined projection operator, it is necessary to impose further restrictions on the sorts of action theories we will use.

3.2 Reasoning in closed and open worlds

So far, we have assumed like Reiter, that $\mathcal{D}_{S_0}$ is any collection of formulas uniform in S_0. Regression reduces the projection problem to that of calculating logical consequences of $\mathcal{D}_{S_0}$. In practice, however, we would like to reduce it to a much more tractable problem than ordinary first-order logical entailment. It it is quite common for applications to assume that $\mathcal{D}_{S_0}$ satisfies additional constraints: domain closure, unique names, and the closed-word assumption [64]. With these, for all practical purposes, $\mathcal{D}_{S_0}$ does behave like a database, and the entailment problem becomes one of database query evaluation. Furthermore, progression is well defined, and behaves like an ordinary database transaction.

Even without using (relational) database technology, the advantage of having a $\mathcal{D}_{S_0}$ constrained in this way is significant. For example, it allows us to use Prolog technology directly to perform projection. For example, to find out if $(\phi \vee \psi)$ holds, it is sufficient to determine if ϕ holds or if ψ holds; to find out if $\neg\phi$ holds, it is sufficient to determine if ϕ does not hold (using negation as failure), and so on. None of these are possible with an unconstrained $\mathcal{D}_{S_0}$.

This comes at a price, however. The unique name, domain closure and closed-world assumptions amount to assuming that we have *complete knowledge* about S_0: anytime we cannot infer that ϕ holds, it will be because we are inferring that $\neg\phi$ holds. We will never have the status of ϕ undecided.

This is obviously a very strong assumption in a cognitive robotic setting, where it is quite natural to assume that a robot will not know everything there is to know about its world. Indeed we would expect that a cognitive robot might start with incomplete knowledge, and only acquire the information it needs by actively *sensing* its environment as necessary.

A proposal for modifying Reiter's proposal for the projection problem along these lines was made by de Giacomo *et al* [15]. They show that a modified version of regression can be made to work with sensing information. They also consider how closed-world reasoning can be used in an open world using what they call *just-in-time queries*. In a nutshell, they require that queries be evaluated only in situations where enough sensing has taken place to give complete information about the query. Overall, the knowledge can be incomplete, but it will be locally complete, and allow us to use closed-world techniques.

Another independent proposal for dealing effectively with open-world reasoning is that of Liu and Levesque [51]. (A related proposal is made by Son and Baral [80] and by Amir and Russell [1].) They show that what they call *proper knowledge bases* represent open-world knowledge. They define a form of progression for these knowledge bases that provides an efficient solution to the projection problem that is always logically sound, and under certain circumstances, also logically complete. The restrictions involve the type of

successor-state axioms that appear in the action theory $\mathcal{D}$: they require action theories that are *local-effect* (actions only change the properties of the objects that are parameters of the action) and *context-complete* (either the actions are context-free or there is complete knowledge about the context of the context-dependent ones).

4 High-level control for cognitive robots

As noted earlier, one distinguishing characteristic of the area of cognitive robotics is that the knowledge representation and reasoning are for a particular purpose: the control of robots or agents. We reason about a world that is changing as the result of actions taken by agents *because* we are attempting to decide what to do, what actions to take towards some goal. This is in contrast, for example, to reasoning for the purposes of answering questions or generating explanations.

4.1 Classical planning

Perhaps the clearest case of this application of knowledge representation and reasoning is in *classical planning* [29]. As discussed in the Situation Calculus chapter, we are given an action theory $\mathcal{D}$ of the sort discussed above and a goal formula, $\phi[s]$ that is uniform in some situation variable s. The task is to find a sequence of ground actions terms $\vec{a}$ such that

$$\mathcal{D} \models \phi[do(\vec{a}, S_0)] \wedge Executable(do(\vec{a}, S_0)).$$

Thus, we are looking for a sequence of actions which, according to what we know in $\mathcal{D}$, can be legally executed starting in S_0 and result in a state where ϕ holds.

Think of having a robot, and wanting it to achieve some goal ϕ. Instead of simply programming it directly, we get the robot to use what is known about the initial state of the world and the actions available to figure out what to do to achieve the goal. This has the very desirable effect that if information about the world changes, that is, if we learn something new, or discover that something old was incorrect, it will not be necessary to reprogram the robot. All we need do is revise its beliefs. Using the terminology of Zenon Pylyshyn [62], we have an architecture that is *cognitively penetrable* in that the behaviour of the robot can be altered by simply changing its beliefs about the world.

In practice, very little of the actual research in classical planning is formulated using the situation calculus in this way. Rather, it is expressed in the more restrictive notation of STRIPS [28]. Instead of an action theory, we have an the initial database formulated as a set of atomic formulas (with an implicit closed-world assumption), and a collection of actions formulated as operators on databases, with preconditions and effects characterized by the additions and deletions they would make to a current database. Although STRIPS has a very operational flavour, it is possible to reconstruct its logical basis in the situation calculus [44, 49].

Despite the restrictions imposed by STRIPS, the classical planning task remains extremely difficult. Even in the propositional case (and with complete knowledge about the initial world state), the problem is NP-hard [10]. While many optimizations exist for many special cases, nobody would consider planning as a *practical* way of generating the millions

of action that might be required of a long-lived robot to achieve long-term goals starting from some initial state.

But this is an unreasonable picture anyway. Nobody would expect *people* to deal with their long-term goals by first closing their eyes and computing a sequence of millions of action, and then blindly carrying out the sequence to achieve the goal, even assuming such a sequence were to exist. This is an *offline* view of how to decide what to do. We need to consider a much more *online* view of high-level control, where as actions are taken, new information that is acquired gets to contribute to the decision-making. Instead of planning in advance for all possible long-term contingencies, we need to be able to get a robot to achieve some part of a goal, assess its current situation, and plan for the rest with the new information taken into account.

4.2 High-level offline robot programming

In an attempt to come up with a more flexible sort of control, one of the directions that has proven to be quite fruitful is the *high-level programming* [42] found in languages such as those in the Golog family [43, 17, 16, 66] and variants like FLUX [84]. Virtually all of the high-level control currently considered in cognitive robotics is of this sort. This brings cognitive robotics closer to the area of *agent-oriented programming* or AOP (see [33, 63], for example).[6]

By a high-level program, we mean a program that contains the usual programming features (like sequence, conditional, iteration, recursive procedures, concurrency) and some novel ones:

- the *primitive statements* of the program are the actions that are characterized by an action theory;
- the *tests* in the program are conditions about the world formulated in the underlying knowledge representation language;
- programs may contain *nondeterministic* operations, where a reasoned choice must be made among alternatives.[7]

Instead of planning given a goal, we now consider program execution given a high-level program. In the situation calculus, Levesque *et al* [43] make this precise as follows: they define an operator $Do(\delta, s, s')$ that maps any high-level program δ into a formula of the situation calculus with two free variables s and s'. Intuitively $Do(\delta, s, s')$ is intended to say that if program δ starts in situation s, one of the situations it may legally terminate in (since the program need not be deterministic) is s'. This is defined inductively on the structure of the program:

$$\text{Primitive action:} \quad Do(A, s, s') \stackrel{\text{def}}{=} Poss(A, s) \wedge s' = do(A, s);$$

$$\text{Test:} \quad Do(\phi?, s, s') \stackrel{\text{def}}{=} \phi[s] \wedge s' = s;$$

[6] This is perhaps a difference of emphasis: cognitive robotics tends to emphasize the robotic interaction with the world, whereas AOP tends to emphasize the mental state of the agent executing the program.

[7] In many applications, we can preserve the effectiveness of an essentially deterministic situation calculus by pushing the nondeterminism into the programming.

Sequence: $Do(\delta_1; \delta_2,\ s,\ s') \stackrel{\text{def}}{=} \exists s''.\, Do(\delta_1,\ s,\ s'') \wedge Do(\delta_2,\ s'',\ s)$;

Nondeterministic branch: $Do(\delta_1 | \delta_2,\ s,\ s') \stackrel{\text{def}}{=} Do(\delta_1,\ s,\ s') \vee Do(\delta_2,\ s,\ s')$;

Nondeterministic value: $Do(\pi x.\ \delta,\ s,\ s') \stackrel{\text{def}}{=} \exists x.\, Do(\delta, s, s')$;

Nondeterministic iteration: $Do(\delta^*,\ s,\ s') \stackrel{\text{def}}{=}$
$\forall P[\forall s_1 P(s_1, s_1) \wedge \forall s_1 s_2 s_3 (P(s_1, s_2) \wedge Do(\delta, s_2, s_3) \supset P(s_1, s_3)) \supset P(s, s')]$.

Other programming common constructs can be defined in terms of these:

if ϕ **then** δ_1 **else** $\delta_2 \stackrel{\text{def}}{=} (\phi?; \delta_1) \,|\, (\neg\phi?; \delta_2)$;

while ϕ **do** $\delta \stackrel{\text{def}}{=} (\phi?; \delta)^*; \neg\phi?$.

The *offline high-level program execution task* then is the following: given a high-level program δ find a sequence of actions $\vec{a}$ such that

$$\mathcal{D} \models Do(\delta, S_0, do(\vec{a}, S_0)).$$

As with planning, we solve this task and then give the resulting action sequence to the robot for execution.

While this is still completely offline like planning, it does allow for far more flexibility in the specification of behaviour. Consider, for example, a high-level program like the following

$$A_1\ ;\ A_2\ ;\ A_3\ ;\ \ldots\ A_n\ ;\ \phi?$$

where each A_i is a primitive action and ϕ is some condition. This program can only be executed in one way, that is, by performing the A_i in sequence and then confirming that ϕ holds in the final state (or fail otherwise). We would naturally expect that solving the execution task for this program would be trivial, even if n were large, since the program already contains the answer. At the other extreme, consider a program like the following:

while $\neg\phi$ **do** $\pi a.\ a$

This is a very nondeterministic program. It says: while ϕ is false, pick an action a and do it. A correct execution of this program is a sequence of actions that can be legally executed and such that ϕ holds in the final state. But finding such a sequence is precisely the planning task for ϕ. So the execution task for this program is no different than the general planning task. However, it is between these two extremes that we can see advantages over planning. Consider this variant:

while $\neg\phi$ **do** $\pi a.\ Acceptable(a)?\ ;\ a$

In this case, we have modified the previous program to include a test that the nondeterministically selected action a must satisfy. Assuming we have appropriate domain-dependent knowledge (represented in $\mathcal{D}$) about this *Acceptable* predicate, we can constrain the planning choices at each stage anyway we like, such as in the forward filtering of [2]. Similarly, we can generalize the first example as in the following:

$$A_1\ ;\ A_2\ ;\ A_3\ ;\ [\textbf{while}\ \neg\psi\ \textbf{do}\ \pi a.a]\ ;\ A_4\ ;\ (A_5 \,|\, B_5)\ ;\ \phi?.$$

In this case, we begin the same way, but then we must solve a (presumably easier) subplanning problem to achieve ψ, then perform A_4, followed by either A_5 or B_5 as appropriate. In nutshell, what we see here is that the high-level program can provide as much or as little procedural guidance as deemed necessary for high-level robot control.

This strategy has proven to be very effective. Among some of the applications built in this way, we mention an automated banking agent that involved a 40-page Golog program [67]. This is an example of high-level specification that would have been completely infeasible formulated as a planning problem.

When a program contains nondeterministic actions, all that matters about the actual choices is that they lead to a successful execution of the entire program. There is no reason to prefer one execution over another. However, real decision making often involves determining which choices are *better* than others. One way to address this issue is to attach numerical *rewards* to situations. Consider, for example, a robot whose only job is to collect objects, but with a preference for red ones. We might use the following successor state axiom for *reward*:

$$\begin{aligned} \mathit{reward}(do(a,s)) = r \;\equiv\; & \\ & \exists x(a = \mathit{pickup}(x) \wedge \mathit{Colour}(x, \mathit{red}, s) \wedge r = \mathit{reward}(s) + 10) \quad \vee \\ & \exists x(a = \mathit{pickup}(x) \wedge \neg \mathit{Colour}(x, \mathit{red}, s) \wedge r = \mathit{reward}(s) + 5) \quad \vee \\ & \neg\exists x(a = \mathit{pickup}(x) \wedge r = \mathit{reward}(s)). \end{aligned}$$

The operator $\mathit{Do}(\delta, s, s')$ introduced above is then replaced by $\mathit{BestDo}(\delta, s, s')$ which selects sequences of actions that maximize accumulated reward. Note that, in the above example, this does not necessarily mean that the robot will always pick up a red object if one is available, as even higher rewards may be unattainable if a red object is picked up now. When combining the idea of maximizing rewards with probabilistic actions, we obtain a decision-theoretic version of Golog, which was first proposed in [8].

4.3 High-level online robot programming

The version of high-level programming we have considered so far has been *offline*. A more online version is considered by de Giacomo *et al* [16, 69]. Instead of using *Do* to define the complete execution of a program, they consider the single-step method first-used to define the offline execution of ConGolog [17]. This is done in terms of two predicates, $\mathit{Final}(\delta, s)$, and $\mathit{Trans}(\delta, s, \delta', s')$. Intuitively, $\mathit{Final}(\delta, s)$ holds when program δ can legally terminate in situation s, and $\mathit{Trans}(\delta, s, \delta', s')$ holds when program δ can legally take one step resulting in situation s', with δ' remaining to be executed. It is then possible to redefine the *Do* in terms of these two predicates:

$$\mathit{Do}(\delta, s, s') \stackrel{\text{def}}{=} \exists\delta'(\mathit{Trans}^*(\delta, S_0, \delta', s') \wedge \mathit{Final}(\delta', s')),$$

where Trans^* is defined as the reflexive transitive closure of *Trans*.[8]

Now imagine that we started with some program δ_0 in S_0, and that at some later point we have executed certain actions $a_1, \ldots a_k$, and that we have obtained sensing results $r_1, \ldots r_k$ from them, with program δ remaining to be executed. The *online high-level program execution task* then is to find out what to do next, defined by:

[8] Much of the work with *Trans* and *Final* requires quantifying over and therefore reifying programs. Some care is required here to ensure consistency since programs may contain formulas in them. See [17] for details.

- stop, if $\mathcal{D} \cup \mathit{Sensed}(\vec{a}, \vec{r}, S_0) \models \mathit{Final}(\delta, do(\vec{a}, S_0))$;
- return the remaining program δ', if

  $$\mathcal{D} \cup \mathit{Sensed}(\vec{a}, \vec{r}, S_0) \models \mathit{Trans}(\delta, do(\vec{a}, S_0), \delta', do(\vec{a}, S_0)),$$

  and no action is required in this step;
- return action b and δ', if

  $$\mathcal{D} \cup \mathit{Sensed}(\vec{a}, \vec{r}, S_0) \models \mathit{Trans}(\delta, do(\vec{a}, S_0), \delta', do(b, do(\vec{a}, S_0))).$$

So the online version of program execution uses the sensing information that has been accumulated so far to decide if it should terminate, take a step of the program with no action required, or take a step with a single action required. In the case that an action is required, the robot can be instructed to perform the action, gather any sensing information this provides, and the online execution process iterates.

The online execution of a high-level program has the advantage of not requiring a reasoner to determine a lengthy course of action, requiring perhaps millions of actions, before executing the first step in the world. It also gets to use the sensing information provided by the first n actions performed so far in deciding what the $(n+1)$ action should be. On the other hand, once an action has been executed in the world, there may be no way of backtracking if it is later found out that a nondeterministic choice was resolved incorrectly. In other words, an online execution of a program may fail where an offline execution would succeed.

To deal with this issue, de Giacomo *et al* propose a new programming construct, a *search operator*. The idea is that given any program δ the program $\Sigma(\delta)$ executes online just like δ does offline. In other words, before taking any action, it first ensures using offline reasoning that this step can be followed successfully by the rest of δ. More precisely, we have that

$$\mathit{Trans}(\Sigma(\delta), s, \Sigma(\delta'), s') \equiv \mathit{Trans}(\delta, s, \delta', s') \wedge \exists s^*.\mathit{Do}(\delta', s', s^*).$$

If δ is the entire program under consideration, $\Sigma(\delta)$ emulates complete offline execution. But consider $[\delta_1 ; \delta_2]$. The execution of $\Sigma([\delta_1 ; \delta_2])$ would make any choice in δ_1 depend on the ability to successfully complete δ_2. But $[\Sigma(\delta_1) ; \delta_2]$ would allow the execution of the two pieces to be done separately: it would be necessary to ensure the successful completion of δ_1 before taking any steps, but consideration of δ_2 is deferred. If we imagine, for example, that δ_2 is a large high-level program, with hundreds of pages of code, perhaps containing Σ operators of its own, this can make the difference between a scheme that is practical and one that is only of theoretical interest.

The idea of interleaving execution and search has also been applied to decision-theoretic Golog [81, 21]. Here, instead of just searching for a successful execution of a sub-program, an optimal sub-plan is generated which maximizes the expected accumulated reward.

Being able to search still raises the question of how much offline reasoning should be performed in an online system. The more offline reasoning we do, the safer the execution will be, as we get to look further into the future in deciding what choices to make now. On the other hand, in spending time doing this reasoning, we are detached from the world and will not be as responsive. This issue is very clearly evident in time-critical applications such

as robot soccer [21] where there is very little time between action choices to contemplate the future. Sardina has cast this problem as the choice between deliberation and reactivity [68], and see also [6].

Another issue arises in this setting is the form of the offline reasoning. Since an online system allows for a robot to acquire information during execution (via sensing actions, or passive sensors, or exogenous events), how should the robot deal with this during offline deliberation [12]. The simplest possibility is to say that it ignores any such information in the plan for the future that it is constructing. A more sophisticated approach would have it construct a plan that would prescribe different behaviour depending on the information acquired during executing. This is *conditional planning* (see, for example, [7, 56]) and one form of this has been incorporated in high-level execution by Lakemeyer [37]. Another possibility is to attempt to *simulate* what will happen external to the robot, and use this information during the deliberation [40]. In [31], this idea is taken even further: at deliberation time a robot uses, for example, a model of its navigation system by computing, say, piece-wise linear approximations of its trajectory; at execution time, this model is then replaced by the real navigation system, which provides position updates as exogenous actions.

Another issues arises whenever a robot performs at least some amount of lookahead in deciding what to do. What should the robot do when the world (as determined by its sensors) does not conform to its predictions (as determined by its action theory)? First steps in logically formalizing this possibility were taken by de Giacomo *et al* [18] in what they call *execution monitoring*. In [21], a simple form of execution monitoring is implemented for soccer-playing robots. Here, the assumptions made by the decision-theoretic planner are explicitly encoded in the generated plan. During execution, these assumptions are re-evaluated against the current world model and, in case of a disagreement, the plan is discarded and a new one generated. See also [26, 34, 22, 23, 24] for related approaches.

5 Conclusion

Cognitive robotics is a reply to the criticism that knowledge representation and reasoning has been overly concerned with reasoning in the abstract and not concerned enough with the dynamic world of an embodied agent. It attempts to address the sort of representation and reasoning problems an autonomous robot would face in trying to decide what to do. In many ways, it has only scratched the surface of the issues that need to be dealt with.

A number of cognitive robotic systems have been implemented on a variety of robotic platforms, using the sort of ideas discussed in this chapter, based either on the situation calculus or on one of the other related knowledge representation formalisms. For a sampling of these systems, see [14, 13, 5, 74, 21, 21, 11, 25]. Perhaps the most impressive demonstration to date was that of the museum tour-guide robot reported in [9].

A fundamental question in the area of cognitive robotics (that Reiter had begun to examine) is the relationship between pure logical representations of incomplete knowledge and the more numerical measures of uncertainty. A start in this direction is the work on the stochastic situation calculus [66] as well as that on noisy sensors and effectors and decision-theoretic Golog, noted above.

On an even broader scale, a much tighter coupling of the high-level control program and

other parts of a robot's software, like mapping and localization, or even vision, is called for. For example, when localization fails and a robot gets lost, it should be possible to use high-level control to do a reasoned failure recovery. Making progress along these lines requires a deep understanding of both cognitive and more traditional robotics, and should help to reduce the gap that currently exists between the two research communities.

References

[1] E. Amir and S. Russell, Logical filtering. *Proc. of the IJCAI-03 Conference*, pages 75–82, Acapulco, 2003.

[2] F. Bacchus and F. Kabanza, Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[3] F., Bacchus, J. Halpern, and H. Levesque, Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence*, 111, 1999, 171–208.

[4] C. Baral, Reasoning about actions: non-deterministic effects, constraints, and qualification. *Proc. of the IJCAI-95 Conference*, 2017–2026, Montreal, 1995.

[5] C. Baral, L. Floriano, A. Hardesty, D. Morales, M. Nogueira, T. C. Son, From theory to practice: the UTEP robot in the AAAI 96 and AAAI 97 robot contests. *Proc. of the Agents-98 Conference,* 32–38, 1998.

[6] C. Baral and T. Son, Relating theories of actions and reactive control. *Electronic Transactions of Artificial Intelligence*, 2(3-4):211-271, 1998.

[7] P. Bertoli, A. Cimatti, M. Roveri, P. Traverso, Planning in nondeterministic domains under partial observability via symbolic model checking. *Proc. of the IJCAI-01 Conference*, 473–478, Seattle, 2001.

[8] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun, Decision-theoretic, high-level agent programming in the situation calculus. *Proc. of the AAAI-00 Conference*, pages 355–362, 2000.

[9] W. Burgard, A. B. Cremers, D. Fox, D Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, S. Thrun, Experiences with an interactive museum tour-guide robot. *Artificial Intellelligence* 114(1-2):3-55, 1999.

[10] T. Bylander, The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[11] A. Carbone, A. Finzi, A. Orlandini, F. Pirri, and G. Ugazio, Augmenting situation awareness via model-based control in rescue robots. *Proc. of IROS-2005 Conference* Edmonton, Canada, 2005.

[12] M. Dastani, F. de Boer, F. Dignum, W. van der Hoek, M. Kroese, J.-J. Meyer, Programming the deliberation cycle of cognitive robots. *Proc. of the 3rd International Cognitive Robotics Workshop*, Edmonton, 2002.

[13] G. de Giacomo, L. Iocchi, D. Nardi, R. Rosati, Moving a robot: the KR & R approach at work. *Proc. of the KR-96 Conference*, 198–209, 1996.

[14] G. de Giacomo, L. Iocchi, D. Nardi, R. Rosati, Planning with sensing for a mobile robot. *Proc. of the ECP-97 Conference*,Toulouse, France. 1997.

[15] G. de Giacomo, and H. Levesque, Projection using regression and sensors. *Proc. of the IJCAI-99 Conference*, Stockholm, Sweden, August 1999, 160–165.

[16] G. de Giacomo, Y. Lespérance, H. Levesque, and S. Sardiña, On the semantics of deliberation in Indigolog. *Annals of Mathematics and Artificial Intelligence*, 41, 2–4, 2004, 259–299.

[17] G. de Giacomo, Y. Lespérance, and H. Levesque, ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121, 2000, 109–169.

[18] G. de Giacomo, R. Reiter, M. Soutchanski, Execution Monitoring of High-Level Robot Programs. *Proc. of the KR-98 Conference*, Trento Italy, 1998.

[19] R. Demolombe, R. and M Pozos Parra, A simple and tractable extension of situation calculus to epistemic logic. *Proc. of the ISMIS-2000 Conference*, 515–524, 2000.

[20] P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and K. Wiklund, The WITAS Unmanned Aerial Vehicle Project. *Proc. ECAI-00*, Berlin, 747–755, 2000.

[21] A. Ferrein, C. Fritz, and G. Lakemeyer. On-line decision-theoretic Golog for unpredictable domains. *Proc. of 27th German Conference on AI*, 322–336, 2004.

[22] A. Finzi and F. Pirri, Diagnosing failures and predicting safe runs in robot control. *Proc. of the Commonsense 2001 Conference,* 105–113. New York, 2001.

[23] A. Finzi and F. Pirri, Combining probabilities, failures and safety in robot control. *Proc. of the IJCAI-01 Conference*, Seattle, August 2001.

[24] A. Finzi and F. Pirri, Representing flexible temporal behaviors in the situation calculus. *Proc. of the IJCAI-05 Conference*, 436-441, 2005.

[25] A. Finzi, F. Pirri, M. Pirrone, and M. Romano, Autonomous mobile manipulators managing perception and failures. *Proc. of the Agents-01 Conference,* 196–201, Montreal 2001.

[26] M. Fichtner, A. Großmann, M. Thielscher, Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57, 371–392, 2003.

[27] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain and H. Turner, Nonmonotonic causal theories, *Artificial Intelligence,* 153:49–104, 2004.

[28] R. Fikes and N. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[29] M. Ghallab, D, Nau, and P. Traverso, *Automated Planning: Theory and Practice.* Morgan Kaufmann, 2004.

[30] H. Grosskreutz and G. Lakemeyer. Turning high-level plans into robot programs in uncertain domains. In Werner Horn, editor, *Proc. of the ECAI-2000 Conference*, pages 548–552, 2000.

[31] H. Grosskreutz and G. Lakemeyer, ccGolog: An action language with continuous change. *Logic Journal of the IGPL,* Oxford University Press, 2003.

[32] H. Grosskreutz and G. Lakemeyer, On-line execution of cc-Golog plans. *Proc. of the IJCAI-01 Conference*, 12–18, 2001.

[33] K. Hindriks, F. de Boer, W. van der Hoek, J.-J. Ch. Meyer, A formal semantics for an abstract agent programming language. *Proc. of the ATAL-97 Conference*, June 1998.

[34] K. Hindriks, F. de Boer, W. van der Hoek, J.-J. Ch. Meyer, Failure, monitoring and recovery in the agent language 3APL. *Proc. of the AAAI-98 Fall Symp. on Cognitive Robotics*, 68–75, 1998.

[35] J. Hintikka,, *Knowledge and Belief.* Cornell University Press, Ithaca, 1962.

[36] W. van der Hoek, J.J. Meyer, B. Linder, On agents that have the ability to choose. *Studia logica*, 66(1), 79-119, 2000.

[37] G. Lakemeyer, On sensing and off-line interpreting in GOLOG. In *Logical Foundations for Cognitive Agents, Contributions in Honor of Ray Reiter,* Springer, Berlin, 173-187, 1999.

[38] G. Lakemeyer, Relevance from an epistemic perspective, *Artificial Intelligence,* 97(1-2):137-167, 1997.

[39] Y. Lespérance, H. Levesque, F. Lin, R. Scherl, Ability and knowing how in the situation calculus. *Studia Logica*, 66, 165–186, October 2000.

[40] Y. Lespérance and H.-K. Ng, Integrating planning into reactive high-level robot programs. *Proc. of the Second International Cognitive Robotics Workshop,* Berlin, Germany, 49–54, 2000.

[41] H. Levesque, What is planning in the presence of sensing? *Proc. of AAAI-96 Conference*, Portland, OR, Aug. 1996, 1139–1146.

[42] H. Levesque and R. Reiter, Beyond planning. *AAAI Spring Symposium on Integrating Robotics Research*, Working notes, Palo Alto, CA, March 1998.

[43] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl, GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.

[44] V. Lifschitz, On the semantics of STRIPS. *Proc. of the 1986 Workshop Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, 1987.

[45] F. Lin, Embracing causality in specifying the indirect effects of actions. *Proc. of the IJCAI-95 Conference*, pages 1985–1991. Montreal, 1995.

[46] F. Lin and R. Reiter. How to progress a database II: The STRIPS connection. *Proc. the IJCAI-95 Conference*, 2001–2007, 1995.

[47] F. Lin and R. Reiter, State constraints revisited. *Journal of Logic and Computation,* 4(5):655-678, 1994.

[48] F. Lin and R. Reiter. Forget it! *Proc. of the AAAI Fall Symposium on Relevance,* New Orleans, USA, November 1994.

[49] F. Lin, R. Reiter, How to progress a database. *Artificial Intelligence*, 92(1–2):131–167, 1997.

[50] B. Linder, W. van der Hoek, J.J. Meyer, Formalizing motivational attitudes of agents: On preferences, goals and commitments. *Proc. of the ATAL-96 Conference*, 17–32, Berlin, 1996.

[51] Y. Liu, H. Levesque, Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. *Proc. of the IJCAI-05 Conference*, Edinburgh, August 2005.

[52] J. McCarthy and P. Hayes, Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, pages 463–502. University of Edinburgh Press, 1969.

[53] D. McDermott, Robot planning. *AI Magazine* 13(2):55–79, 1992.

[54] R. Moore, A formal theory of knowledge and action. *Formal Theories of the Commonsense World*, Ablex, Norwood, NJ, 319–358, 1985.

[55] N. Nilsson, Shakey the robot. SRI Technical report, 1984.

[56] F. Bacchus and R. Petrick, Modeling an agent's incomplete knowledge during planning and execution. *Proc. of the KR-98 Conference*, Trento, Italy, 1998.

[57] M. Fox and D. Long, PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research*, 20:61–124, 2003.

[58] J. Pinto, Integrating discrete and continuous change in a logical framework. *Computational Intelligence, 14(1)*, 1997.

[59] F. Pirri and A. Finzi, An approach to perception in theory of actions: Part I. *Electronic Transaction on Artificial Intelligence,* 3(41):19–61, 1999.

[60] F. Pirri and M. Romano, A situation-Bayes view of object recognition based on symgeons. *Proc. of the Third International Cognitive Robotics Workshop*, Edmonton, 2002.

[61] F. F. Ingrand, R. Chatila, R. Alami and F. Robert, PRS: A high level supervision and control language for autonomous mobile robots. *Proc. Int. Conf. on Robotics and Automation*, 1996.

[62] Z. Pylyshyn, *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press, Cambridge, Massachusetts, 1984.

[63] A. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language. *Agents Breaking Away*, Springer-Verlag, 1996.

[64] R. Reiter, On closed world data bases. *Logic and Databases*, pages 55–76. Plenum Press, New York, 1987.

[65] R. Reiter, The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, New York, 1991.

[66] R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, Massachusetts, 2001.

[67] S. Ruman, *GOLOG as an Agent-Programming Language: Experiments in Developing Banking Applications.* M. Sc., Dept. of Computer Science, University of Toronto, January 1996.

[68] S. Sardina, *Deliberation in agent programming languages.* Ph. D. thesis, Dept. of Computer Science, University of Toronto, June 2005.

[69] S. Sardiña, *Indigolog: Execution of guarded action theories.* M. Sc. Thesis, Dept. of Computer Science, University of Toronto, April 2000.

[70] M. P. Shanahan, *Solving the Frame Problem,* MIT Press, 1997.

[71] M. P. Shanahan, A logical account of perception incorporating feedback and expectation. *Proc. of the KR-02 Conference*, 3–13, 2002

[72] M. P. Shanahan, Representing continuous change in the event calculus. *Proc. of the ECAI-90 Conference*, 1990.

[73] M. P. Shanahan, Noise and the Common Sense Informatic Situation for a Mobile Robot, *Proc. of the AAAI-96 Conference*, 1098–1103, 1996.

[74] M.P.Shanahan, Reinventing Shakey. In *Logic-Based Artificial Intelligence,* Jack Minker (Ed.), Kluwer Academic, 233–253, 2000.

[75] M. P. Shanahan, M. Witkowski, High-Level Robot Control Through Logic. *Proc. of the ATAL-2000 Conference*, 104–121, 2001.

[76] S. Shapiro, M. Pagnucco, Y. Lespérance, H. Levesque, Iterated belief change in the situation calculus. *Proc. of the KR-2000 Conference*, Breckenridge CO, April 2000, 527–538.

[77] S. Shapiro and M. Pagnucco, Iterated belief change and exogenous actions in the situation calculus. *Proc. of the ECAI-04 Conference*, 878–882, 2004.

[78] S. Shapiro, Y. Lesperance, H. Levesque, Goal change. *Proc. of the IJCAI-05 Conference*, Edinburgh, August 2005.

[79] R. Scherl, H. Levesque, Knowledge, action, and the frame problem. *Artificial Intelligence*, 144, 2003, 1–39.

[80] T. Son and C. Baral, Formalizing sensing actions – A transition function based approach. *Artificial Intelligence*, 125(1–2):19–91, 2001.

[81] M. Soutchanski, An on-line decision-theoretic golog interpreter. *Proc. of the IJCAI-01 Conference,* Seattle, Washington, 2001.

[82] M. Thielscher, Modeling actions with ramifications in nondeterministic, concurrent, and continuous domains - and a case study. *Proc. of the AAAI-00 Conference* 497-502, 2000.

[83] M. Thielscher, Representing the knowledge of a robot. *Proc. of the KR-2000 Conference*, Breckenridge, 109-120, 2000.

[84] M. Thielscher, FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 5(4–5):533–565, 2005.

[85] M. Thielscher, *Reasoning Robots: The Art and Science of Programming Robotic Agents*. Springer, 2005.

[86] S. Thrun, Robotic mapping: A survey. In G. Lakemeyer and B. Nebel (Eds.) *Exploring Artificial Intelligence in the New Millennium.* Morgan Kaufmann, 2002.