1 Introduction

A theory of action allows us to *specify* in an elaboration tolerant [McC59, MH69] manner the relationship among fluents – physical objects of the world that may change their values or mental objects that encode the mental state of the robot/agent, and the effects of actions on these fluents and *reason* about them.

Using theories of actions we can make *predictions* about the state the world will be in after the execution of a sequence of actions (or even after the execution of a 'program' of actions) in a particular world.

For example, a theory of action should allow us to specify the effect of the action 'shoot' which says: 'shooting causes the turkey to be dead if the gun is loaded'; and the effect of the action 'load' which says: 'loading the gun causes the gun to be loaded'. It should allow us to specify the relationship between the fluents 'dead' and 'alive' which says: 'the turkey is dead if and only if it is not alive'. We should then be able to use the theory of action to reason and predict that if the turkey is initially alive and the gun is unloaded, by loading the gun and then shooting the turkey will be dead.

This ability allows us to *make plans* that will take us to particular kind of worlds. In the above example, the plan to reach the world where the turkey is dead from a world where the turkey is alive and the gun is unloaded is the sequence of actions: 'load' followed by 'shoot'.

Using theories of actions we can also *explain* observations about the state of the world in terms of what actions might have taken place, or what state the world might have been before, or both. For example, if we knew that the turkey was initially alive, the gun was initially loaded, and we observe that the turkey is dead now, we can explain that the action 'shooting' must have taken place. Similarly if we knew that the turkey was initially alive and we observe that it is dead now, and we also observed that 'shooting' took place then we can explain that the gun must have been loaded before shooting took place.

Recently there has been a lot of progress in formulating theories of actions, particularly in progressing from simple and/or restricted theories [FN71] and 'example centered approaches' [HM87] (also papers in [Bro87]) – which were extremely useful, to general and provenly correct theories [GL93, San92, San94, Rei91, LS91, LS95] that incrementally consider various *specification aspects* such as: actions with non-deterministic effects [Bar95, Pin94, KL94, San93], concurrent actions [LS92, BG93, BG97a], narratives [MS94, PR93, BGP97, BGPml, Kar98], actions with duration [Sho86, San89, San93, San94, MS94, Rei96], natural events [Rei96], ramifications and qualifications due to simple and causal constraints [LR94, KL94, Bar95, Lin97, Thi97, MT95, Lin95, Bar95, GL95, San96, GD96], sensing (or knowledge producing) actions [Moo77, Moo79, Moo85, SL93, LTM97, BS97], continuous and hybrid change [San94], etc. Most of the above formalizations define an entailment relationship ($\models$) between the specifications (of effects of actions and relation among objects of the world) and *simple queries* of the form f **after** $a_1, \ldots, a_n$, where f is a fluent (or a set of fluents) and a_i's are actions. The statement that a theory of action D allows us to predict that turkey will be dead after loading followed by shooting can be formally written as:

$$D \models dead \ \textbf{after} \ \ load, shoot$$

Recently more general queries have also been considered where f is generalized to a statement about the evolution of the world and is given as a formula in a language with knowledge and temporal operators [BK96, MLLB96], and simple plans (i.e., sequences of actions) in the simple query are generalized to a conditional plan [Lev96, BS97] or a complex plan [LLL$^+$94, San94, LRL$^+$97]. The books [Sha97, San94, Rei98], the special issues [Geo94, Lif97], and the workshop proceedings [BGD$^+$95, Bar96] contain additional pointers to recent research in this area.

Theories of actions can be used to model *deliberative* agents. In case of a *static world* theories that allow only observation about the initial state[1] is sufficient. The architecture of the agent then consists of (i) *making observations*, (ii) using the action theory to *construct a plan* to achieve the goal, and (iii) *executing the plan.* In case of a *dynamic world* where other agents may change the world we need theories that allow observations as the world evolves [BGP97]. With such a theory we can modify the earlier architecture so that in step (ii) plans are constructed from the *current state*, and in step (iii) only a part of the plan is executed and the agent repeatedly executes Step (i) and the modified steps (ii) and (iii) until the goal is satisfied. Such an architecture is discussed in [BGP97].

But the above deliberative architecture requires *on-line planning*, which is in general time consuming even for very restricted theories of action [ENS95]. This makes it impractical for many kinds of autonomous agents, particularly the ones that are in a rapidly changing environment and that need *to react* to the changing environment in real time. This includes mobile robots and shuttle control agents.

For such agents a viable alternative is to consider control architectures that are *reactive* in nature. A simple reactive control module is a collection of control rules of the form:

$$\textbf{if } p_1, \ldots, p_n \textbf{ then } a$$

Intuitively the above rule means that if the agent believes (based on its sensor readings, its prior knowledge about the world, and its mental state) $p_1, \ldots, p_n$ (the LHS) to be true in a situation then it must execute the action a (the RHS). The agent continuously senses (or makes observations) and looks for a control rule (in its control module) whose LHS is true and executes its RHS. Reactive control modules were discussed in the domain of controlling mobile robots in [Fir87, GL87, Bro86] and in the papers in the collection [Mae91a]. They were also discussed in [Dru86, Dru89, DB90, DBS94, Nil94, Sch95, Sch89b] for other domains. (Some of the other research regarding the role of reactivity in planning and execution is described in [LHM91, Mus94, RLU90, McD90, Mit90].)

The main advantage of the reactive approach is that after sensing (or making observations) the agent need not spend a lot of time in deliberating or constructing a plan; rather it just needs to do a 'table-lookup' to determine what actions to take. The later is much less time consuming and is well suited for agents that need to respond quickly to the changing environment.

In the earlier paragraphs we described the relationship between *deliberative control* and theories of actions and discussed how theories of actions can be used in formulating a deliberative control architecture. Considering that

[1] Throughout this paper we use the terms 'state' and 'situation' interchangeably.

the reactive modules also use actions (in the RHS of the control rules) and fluents (in the LHS of the rules), a question that comes up is: Is there some relationship between action theories and reactive control; especially in the presence of exogenous actions[2] and when the control is not a universal plan? Although some attempts were made in [SKR95, KR91, Dru89, Sch89b], which we will further discuss in Section 10, the relationship between them is not well established. *The goal of this paper is to formalize this relationship.*

As a result we achieve the following in this paper:

- We formulate the notion of a 'reactive control module' being 'correct' with respect to achieving (also, maintaining) a goal, a set of initial states, a theory of action of the agent, and a theory of action about the exogenous actions.
- We develop sufficiency conditions that guarantee the correctness of individual control rules and a control module as a whole and use it to develop an algorithm that generates 'correct' control modules.
- We extend the previous two results to the case where the agent is able to perform 'sensing' actions and such actions are allowed in the control module.

1.1 The intuitive relation between action theories and reactive control

Intuitively, a reactive module can be thought of as a compiled table obtained from action theories.

To elaborate on our intuition, let us consider an analogy with language theory and parsers. The language theory corresponds to the theory of actions and the parsing table and the parser correspond to the control rules and the control architecture, respectively. The parser needs to parse quickly and hence uses a parsing table. But the parsing table is constructed using the theory of languages and it can be formally shown that a particular parsing table will indeed correctly parse with respect to a particular language theory. The reactive agent in its need to act quickly needs to use control rules, and as in the case of parsing we should be able to: formally show the correctness of a control module - meaning, we should be able to formally prove that an agent following a particular control module will indeed achieve a certain goal (or maintain a certain status); and perhaps under some conditions[3], be able to automatically construct 'correct' control modules for a given goal from an action theory.
As mentioned before, these are the main goals of this paper.

One way to break down the above goals is to formulate the correctness of individual control rules. Intuitively, the action that a particular control rule directs an agent to do in a certain situation is one of the actions that *leads*

[2] Often the term 'exogenous event' or 'natural event' is used instead of the term 'exogenous action'. We do not use the term 'natural' as our exogenous actions may consist of actions of other agents, besides nature. We use the term 'exogenous action' instead of 'exogenous event' to avoid talking about an event theory separately. By referring to them as actions, we can just say action theory and do not have to say action theory for robot's actions and event theory for exogenous events.

[3] These conditions are analogous to the restriction on the class of languages for which we can construct a parsing table.

to the goal based on the theory about the actions. The notion of 'an action leading to a goal' was introduced in [KR91], but was never formalized. The sufficiency conditions in Section 4.1 give a formalization of this notion. *Our formalization states that, an action a leads to the goal g, from a situation s, if a is the first action in a minimal cost*[4] plan *that achieves g from s.*

Recall, the role of *making plans* in the deliberative architecture discussed earlier. In the process of achieving our main goal we have now stumbled upon the connection between *reactivity* and *deliberative reasoning. Deliberative reasoning can be used off-line, where processing time is not that critical, to generate reactive control rules which will be used on-line to react in real-time.* This leads us to the motto behind our work:

> OFF-LINE DELIBERATIONS FOR ON-LINE REACTIVITY[5]
> – for commonly encountered situations.

In fact, in our opinion, this has a lot of similarity with human behavior. A significant amount[6] of our actions are reactive in nature, and most time-critical actions are reactive in nature. Moreover, most of our reactive actions are either compiled using deliberative reasoning or told to us with justifications on why it is right. Let us take the example of the case of driving an automobile. When we first start learning to drive, our driving instructor tells us the function of different control elements of the car (i.e., effects of various actions that we can do while on the controls of the car), he tells us about the driving environment, and not only what to do in a situation but also why. The 'what' part are reactive control rules that have been compiled over the ages since people have been driving. The 'why' part is usually justification of the usefulness of the control rules. Of course, we ourselves, develop our own additional control rules based on (i.e., by compiling) our knowledge about the actions we can take, and their effects, some of which we may not be told by our instructor, or we may not have read from the driving handbook, but we have learned[7] from our own experience.

Another example is the concept of a fire drill. When there is a fire, we do not have time to deliberate and figure out our escape route. Hence, the need for off-line deliberations about escape route in case of a fire and constructing reactive rules that need to be executed if the need arises.[8] Another example is reactive plans constructed (off-line) by city authorities to tackle emergencies.

Although, the whole idea of reactive rules being some kind of compilation of knowledge (in this case, about the actions and their effects) is

---

[4] In Section 4.1 we motivate and elaborate on the intuition behind choosing minimal cost as our criterion.

[5] Moses and Tennenholtz in [MT96] advocate a similar approach with respect to performing off-line reasoning for on-line efficiency in answering queries with respect to a knowledge base. They introduce a formal notion of 'efficient basis' and show that off-line preprocessing can be very effective for query languages that have an efficient basis.

[6] The others can be divided into two categories: semi-reactive and deliberative. Semi-reactive actions are actions that are picked after a small amount of reasoning.

[7] The learning aspect, although very important, is beyond the scope of this paper.

[8] The role of the fire drill is to memorize the reactive rules. Perhaps, since the memory in human being is not a RAM, but more like a neural network, the fire drill is analogous to the training of a neural network so that it memorizes a condition-action table.

probably not new, *what is new in this paper is that we formalize precisely what this compilation is.* Note that, in the past, since this connection was not precise, it resulted in confusion regarding when to be reactive and when to be deliberative. For example, Brooks [Bro91a, Bro91b] suggested that reasoning and high level planning may not be at all necessary in controlling a mobile robot, and this was fairly controversial at that time. But once we understand the connection between being reactive and being deliberative, there is no controversy – Once we compile (off-line, and using deliberation) our knowledge about actions and their effects into reactive rules, we no longer have to reason or do high-level planning on-line, when the mobile robot is in action. Many other [Ark91, KR91, MS91] researchers advocated to combine high level planning and reasoning with reactivity. Their suggestion was to use reactivity at the lower level (such as avoiding obstacles) and use high-level planning at the higher level (such as to decide which room to visit first while doing a task in an office environment). Once the connection between reactivity and deliberativity is clear, depending on the required response time (at different levels) and the space available to store reactive rules, the designer of an autonomous agent can decide on making which levels reactive and which levels deliberative.

This brings us to the issue of space required in storing reactive rules. Earlier we discussed the correctness of an individual control rule. But to formalize the correctness of a control module, we need to worry not only about the individual control rules in it, but also about the set of rules as a whole. Intuitively, an individual control rule tells the agent what to do in situations that matches its LHS. For the correctness of the whole module, we need to consider *a set of plausible states* that the agent may be in and make sure that there is at least one control rule in our module (not necessarily distinct) for each of the plausible state. We only consider a set of plausible states instead of the set of all states to avoid the number of reactive rules from becoming prohibitively large. But since we have exogenous actions, there is a possibility that the agent may reach a state for which it does not have a control rule. In that case it can use deliberation to make plans to reach one of the plausible states, from where it has control rules telling it what to do. Note that this deliberation in most cases would be less time consuming than making a plan to reach the goal. This is analogous to a lost driver trying to reach a known landmark from where he knows how to get to his destination. This may not always work though, and may put the agent in a loop. To avoid getting into a loop the agent needs to be able to recognize failure of actions and reason about them [TSG95].

Finally, viewing reactivity as a compilation obtained by deliberations, provides some insight into the puzzle that reasoning with logics proposed for common-sense reasoning (such as default logic, auto-epistemic logic, circumscription, and declarative logic programming) have been inefficient. The insight is that the inefficient reasoning is to be done off-line (when time is less of a concern) to compile and obtain a look-up table (or perhaps layers of look-up tables) that can be used to react quickly on-line. This explains, the quick reacting ability of people when faced with expected or planned for situations, and the response of, ‘let me think’ when faced with an unplanned situation. And many times, the ‘thinking’ phase goes on for days in the human reasoner.

1.2 Organization of the rest of the paper

The paper is organized as follows. In Section 2 we consider simple control modules and give their operational semantics. We then give some examples of control modules together with the specifications of the actions used in those control modules. In Section 3 we formally characterize the correctness of simple control modules and in the process introduce the concept of *closure* and *unfolding* of control modules. We use a running example to illustrate these concepts. In Section 4 we present sufficiency conditions for the correctness of control modules and use them to present two different algorithms that automatically construct correct control modules. In Section 5 we extend simple control modules to allow sensing actions and to encode conditional plans. In Section 6 we formalize the correctness of the extended control modules and in Section 7 we present sufficiency conditions for these modules. We then use the sufficiency conditions to present an algorithm that constructs such control modules. In section 8 we briefly discuss our view on how an agent should be both reactive and deliberative and how these two aspects are integrated. In section 9 we describe the architecture of our mobile robot entry in AAAI 96 and relate it to the theory of agents developed in the previous sections. In section 10 we relate our research in this paper to earlier work on universal plans, situation control rules, and agent theories and architectures. In section 11 we briefly discuss several future directions to our work and relate the methodology in this paper to the views of Dijkstra and Hoare on programming methodologies in general.

2 Simple Control Modules: preliminaries

In this paper we consider a robot control architecture consisting of hierarchically defined reactive control modules. *Throughout the paper we use the term 'robot' and 'agent' in a generic sense and each subsume both software based agents – such as softbots, and physical robots.* By 'hierarchically' we mean that the action used in the control rule of a control module may itself be defined using another control module. Similar architectures are also suggested in [Sch87, Nil94, Fir92, BKMS95].

In this section we define simple control modules and their operational semantics. In a later section (Section 5) we extend our definition to allow sensing actions.

Definition 2.1 *[Control rules and Control modules]* A *simple control rule* is of the form,

if $p_1, \ldots, p_k$ **then** $a_1, \ldots, a_l$

where $p_1, \ldots, p_k$ are fluent literals and $a_1, \ldots, a_l$ are actions.

A *termination control rule* is of the form

if $p_1, \ldots, p_k$ **then** $HALT$,

and a *suspension control rule* is of the form

if $p_1, \ldots, p_k$ **then** $SUSPEND$,

A control rule is a simple control rule, a termination control rule, or a suspension control rule. The part between the **if** and **then** of a control rule is referred to as the LHS of the rule and the part after the **then** is referred to as the RHS of the rule.

A *control module* is defined as a collection of control rules.

Achievement control module, and mixed control modules consist of only simple and termination control rules. A maintenance control module consists of only simple and suspension control rules. □

Intuitively an achievement control module is supposed to make a given goal true, a maintainance control module is suppose to maintain a goal true, and a mixed control module is supposed to make a goal true while maintaining another. Achievement and mixed control modules halt after achieving their goals. Maintainance control modules execute continuously until they are stopped from outside. We now define the operational semantics of simple control modules.

2.1 Operational semantics of simple control modules

A simple control module can be in four different states: *active, suspended, success-terminated*, and *failure-terminated*.

In the active state it continuously executes the following loop: *observe, match, and act*.

In the observe cycle it reads its sensor values and quickly computes and updates the fluent values. Note that although many of the fluents, which we call basic fluents, may directly correspond to sensor values with possible use of thresholds, there may be fluents whose values are derived from the basic fluents. *There may be other fluents which do not correspond to any sensors but encode the mental state of the robot.*

In the match cycle it matches the values of the fluents with the LHS of the rules.

In the act cycle it executes the actions in the RHS of all the rules whose LHS was matched successfully. If there are more than one such rules and the actions in their RHS are different but non-contradicting then it executes them concurrently. If they are contradicting then it uses some priority mechanism (similar to the approach in the subsumption architecture [Bro86]) to decide which ones to execute. If the RHS of the rule is $HALT$ then the control module reaches the *success-terminated* state. If the RHS of the rule is $SUSPEND$ then the control module reaches the *suspended* state.

In the suspended state the sensors are active and any change in the sensor values takes the robot from the *suspended* state to the *active* state.

If in the match cycle no rule is found whose LHS is matched then the control module reaches the *failure-terminated* state.

2.2 Assumptions about simple control modules and their limitations

We have the following assumptions about our robot and the environment it is in. These assumptions play a crucial role in our formulation of correctness of control modules.

1. After each observe step the robot has complete information about each fluents.
2. The internal state of the robot is correct with respect to the world state. (I.e. the modeling of the world and the sensing are perfect.)
3. Actions are duration less.

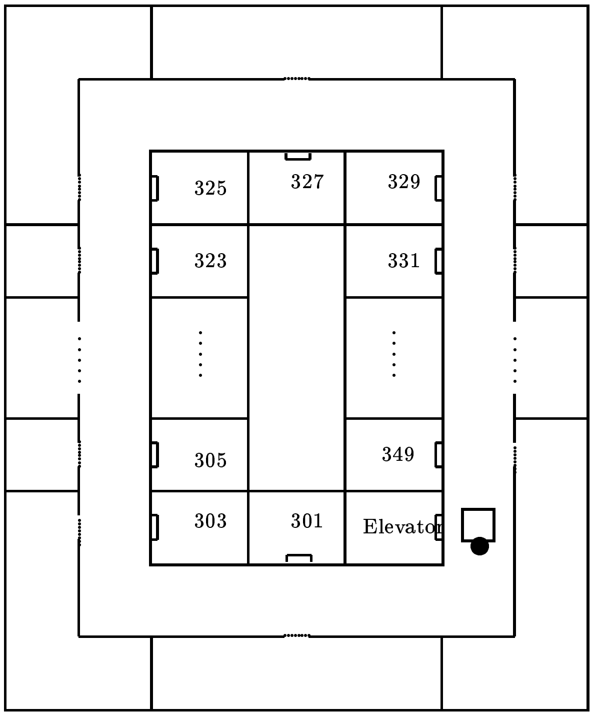

Figure 1: Mobile robot in a simple office floor.

4. Robots actions and the exogenous actions do not overlap or happen at the same time.

5. The control module may have rules only for some states.

The first assumption and the second assumption together mean that the robots internal states reflect world states accurately. This is a limitation of our approach when applied to physical robots at the lowest level of control. The above assumptions are often appropriate for the higher level of control of physical robots and in softbots. (Later in Section 5 we remove the first assumption.) The third assumption is due to the action theories that we will be using for formulating correctness of control modules. Although at first glance they seem to be restrictive, Reiter in [Rei96] shows how actions that take time can be essentially modeled using instantaneous 'start' and 'stop' actions. The viability of the fourth assumption depends on the third assumption and it is well known that by choosing an appropriate granularity parallelism can be modeled using concurrency. The fifth assumption reflects the fact that many control modules may not be universal plans [Sch87] and may only have rules for some of the states. This has consequences when we formalize the correctness of control modules.

2.3 Some Examples of Control Modules

Consider a mobile robot navigating the office floor in Figure 1. Assuming that the robot's sensing mechanism can tell where the robot is located at, in terms of being next to rooms 301 to 349 or next to the elevator, the following control module when executed can take the robot next to the elevator from anywhere in the office floor.

Module : Goto_elevator_1
if $\neg at\_elevator$ **then** $Go\_clockwise\_1room$
if $at\_elevator$ **then** $HALT$ □

In the above control module we have assumed that the robot has an action which when executed takes the robot to the next room in the clockwise direction.

To formally show the correctness of the control module $Goto\_elevator\_1$ we need to specify the effect of the action $Go\_clockwise\_1room$, which is used in the action theory, and also any constraints about the world. For this we will use the high-level syntax of the language $\mathcal{A}$ [GL93] and its successor $\mathcal{AR}$ [KL94].

Specifying effects of the actions in module Goto_elevator_1

$Go\_clockwise\_1room$ **causes** $at\_room(X+2)$
if $at\_room(X), \neg at\_room(349)$
$Go\_clockwise\_1room$ **causes** $at\_elevator$ **if** $at\_room(349)$
$Go\_clockwise\_1room$ **causes** $at\_room(301)$ **if** $at\_elevator$

Specifying constraints about the world corresponding to the module Goto_elevator_1

always $(at\_room(301) \oplus \ldots \oplus at\_room(349) \oplus at\_elevator)$[9]

It should be noted that for mobile robots the effects of actions and the constraints that we specify is from the robot's perspective[10]*. It may not be from the perspective of an impartial observer. The later is usually used in action theories, but we do not use it in robots situated in a physical world. This is to avoid an additional mapping between the robot's perspective as described by its sensors and the world model.* For other robots/agents that are not situated in a physical world the action theory may be specified from an impartial observers perspective.

Following is another control module to take the robot next to the elevator. This module uses an additional action $Go\_anticlockwise\_1room$, whose effects are also specified below.

Module : Goto_elevator_2

if $at\_room(X), X \geq 325$ **then** $Go\_clockwise\_1room$
if $at\_room(X), X \leq 323$ **then** $Go\_anticlockwise\_1room$
if $at\_elevator$ **then** $HALT$ □

Specifying the effect of the action Go_anticlockwise_1room

$Go\_anticlockwise\_1room$ **causes** $at\_room(X-2)$
if $at\_room(X), \neg at\_room(301)$
$Go\_anticlockwise\_1room$ **causes** $at\_elevator$ **if** $at\_room(301)$
$Go\_anticlockwise\_1room$ **causes** $at\_room(349)$ **if** $at\_elevator$

We can now express the goal of this paper in terms of the above examples. *Our initial goal is* to be able to formally show using the above action theories that both the control modules $Goto\_elevator\_1$ and $Goto\_elevator\_2$ will take the robot next to the elevator. *Moreover* we would like to be able to automatically construct the control modules from the action theory given the goal that the control module should take the robot next to the elevator.

In the above control modules we used the actions $Go\_clockwise\_1room$ and $Go\_anticlockwise\_1room$, and described their effects; but we did not

[9] The symbol $\oplus$ denotes ex-or.

[10] Lespérance and Levesque in [LL95] give a detailed logical account of knowledge from the robot's perspective – termed 'indexical knowledge'.

specify what these actions consists of. Following our hierarchical approach we can further *define* the actions *Go_clockwise_1room* and *Go_anticlockwise_1room* as control modules.

We will now define the control module corresponding to the action *Go_clockwise_1room*. (The control module for the other actions can be similarly defined.) To make our control module simple we make certain assumptions about the environment. We assume that the doors of the outer rooms are painted white and the doors of the inner rooms are painted black. We also assume that the control program that takes the robot out of a room also aligns the robot such that its side faces the door, and when this control module is called for execution the *mental fluent just_started* is assigned the value *true*. Besides fluents that are directly dependent on sensor readings, we have fluents that do not correspond to any sensors but rather encode the state of the robot. Recall that we refer to such fluents as *mental fluents* and in the following control module *just_started* is such a fluent. In general exogenous actions can not directly change the values of mental fluents; they can only be changed by a direct action of the robot.

Module : Go_clockwise_1room

if $white\_door\_on\_rt$ **then** $turn\_180$
if $just\_started, black\_door\_on\_rt$ **then** $go\_forward$
if $just\_started, wall\_on\_rt$ **then** $del\_started$
if $\neg just\_started, wall\_on\_rt$ **then** $go\_forward$
if $corridor\_on\_rt$ **then** $turn\_rt\_90$
if $\neg just\_started, black\_door\_on\_rt$ **then** $HALT$ □

In the above module the action $turn\_180$ turns the robot 180 degrees; the action $go\_forward$ takes the robot forward a certain distance until there is a change in its sensor values regarding what is in its right; the action $del\_started$ falsifies the mental fluent $just\_started$; and the action $turn\_rt\_90$ first makes the robot turn 90 degrees to the right and then makes it go forward until the wall is on its right. Also note that the actions $go\_forward$ and $turn\_rt\_90$ can be further defined by another control module. We now formally specify the action theory that describes the above effects of the actions and also specifies the relationship between certain fluents.

Action theory for the actions in control module Go_clockwise_1room

$turn\_180$ **causes** $black\_door\_on\_rt$ **if** $white\_door\_on\_rt$
$go\_forward$ **causes** $wall\_on\_rt$ **if** $black\_door\_on\_rt$
$go\_forward$ **causes** $black\_door\_on\_rt$ **if** $wall\_on\_rt, \neg app\_corner$
$go\_forward$ **causes** $at\_room(X+2)$
  if $wall\_on\_rt, \neg app\_corner, at\_room(X), \neg at\_room(349)$
$go\_forward$ **causes** $at\_elevator$ **if** $wall\_on\_rt, \neg app\_corner,$
  $at\_room(349)$
$go\_forward$ **causes** $at\_room(301)$ **if** $wall\_on\_rt, \neg app\_corner,$
  $at\_elevator$
$go\_forward$ **causes** $corridor\_on\_rt$ **if** $wall\_on\_rt, app\_corner$
$go\_forward$ **causes** $\neg app\_corner$ **if** $wall\_on\_rt, app\_corner$
$turn\_rt\_90$ **causes** $wall\_on\_rt$ **if** $corridor\_on\_rt$
$del\_started$ **causes** $\neg just\_started$ **if** $wall\_on\_rt$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at\_room(301)$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at\_room(325)$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at\_room(327)$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at\_elevator$

always $(black\_door\_on\_rt \oplus white\_door\_on\_rt$
$\oplus wall\_on\_rt \oplus corridor\_on\_rt)$
always $(at(other)$ iff $(at\_room(303) \oplus \ldots \oplus at\_room(323) \oplus$
$at\_room(329) \oplus \ldots \oplus at\_room(349)))$
always $(at\_room(301) \oplus \ldots \oplus at\_room(349) \oplus at\_elevator)$
always $(app\_corner \Rightarrow \neg at(other))$
always $(corridor\_on\_rt \Rightarrow \neg at(other))$
always $(corridor\_on\_rt \Rightarrow \neg app\_corner)$ □

To complete the flavor of the various kind of control modules that we may have we now give example of a maintainance control module $Maintain\_Siren$ (a similar module is discussed in [JF93]) whose purpose is to maintain the goal $\neg off\_siren$.

Module : Maintain_Siren

if off_siren **then** $turn\_on\_siren$
if $\neg off\_siren$ **then** $SUSPEND$ □

Notice that the above control module does not contain any rule that has HALT in its RHS. This means once the execution of the control module starts it never terminates by itself.

The effect of the action $turn\_on\_siren$ is specified as follows:

$turn\_on\_siren$ **causes** $\neg off\_siren$

In the future sections we will formulate correctness of control modules and show the correctness of the control modules discussed in this section. We will also discuss how to automatically generate such control modules.

2.4 Specifying actions and their effects

Because of assumptions 1 and 2 in Section 2.2 a simple action theory without features such as being able to observe (as in [BGP97]), or having narratives [MS94, PR93], or having knowledge producing actions [Moo85, SL93], is sufficient for formulating correctness of the control module discussed in the previous section.

This is because of the fact that our robot with a reactive control does not reason about its past. It just takes into account the current sensor values and the current mental state of the robot (and possibly some additional fluents) to decide what actions to do next. Also it does not completely rely on its actions and allows the possibility of outside interference. After executing an action it senses again and proceeds from there.

Our action theory has two kinds of actions: one that the robot can perform, and the other that may happen independent of the robot and which is beyond the control of the robot. The second kind of action referred to as exogenous actions may frustrate the robot trying to achieve the goal or may provide the robot with an opportunity. For both kinds we have effect axioms that describe the effect of the actions. Our theory allows us to express values of fluents in particular situations and allows us to reason in the forward direction from that situation. In other words, given values of fluents in situation s, our theory lets us determine if a fluent f is true in the situation $Res(a_n, Res(a_{n-1}, \ldots, Res(a_1, s) \ldots)$.[11] We usually denote this by $\models holds(f, [a_1, \ldots, a_n]s)$ or by $\models f$ **after** $[a_1, \ldots, a_n]$ **at** s. When necessary, an action a_i could be a compound action consisting of concurrent execution of a sequence of basic actions - actions which can not be decomposed further. For example, by $\{[a_{1,1}, \ldots, a_{1,k_1}], \ldots, [a_{m,1}, \ldots, a_{m,k_m}]\}$, we

[11] We often denote this situation by $[a_1, \ldots, a_n]s$.

denote a compound action, whose execution corresponds to the concurrent execution of the sequences of actions $[a_{1,1},\ldots a_{1,k_1}],\ldots,[a_{m,1},\ldots,a_{m,k_m}]$. Compound actions are treated in [BG93, BG97a, LS92, GLR91, ALP94]. While, in [BG93, BG97a, LS92, GLR91] concurrent execution of actions are more like parallel execution, in [ALP94] concurrent execution of actions correspond to concurrent transaction processing in databases.

When we refer to a plan that achieves a goal, the plan consists of only the actions that can be performed by the robot. The other actions are only used to determine states that the robot may be in.

In this paper we do not advocate or consider any particular theory of action. Any theory that has the above mentioned entailment relation, that subscribes to our notion of two different kinds of actions, and can reason about concurrent executions of sequences of actions is suitable[12] for our purpose. Moreover certain simplifications in the class of control modules we allow, such as the LHS of two rules not being simultaneously true, results in simplifying the required action theories. In this particular case, the simplification allows us to use the theory $\mathcal{A}$ [GL93] or $\mathcal{AR}$ [KL94]. In most of the paper we will be using such a theory.

2.5 Valid states during the execution of Go_clockwise_1room

We will be using the module *Go_clockwise_1room* as a running example in the next two sections. As a first step, we will now list all the states the robot may be in while executing this module. At times we will use some abstractions.

From the domain constraint
always $(black\_door\_on\_rt \oplus white\_door\_on\_rt \oplus wall\_on\_rt \oplus corridor\_on\_rt)$
we know that in every state one and only one of the fluents
black_door_on_rt, *white_door_on_rt*, *wall_on_rt*, or *corridor_on_rt* must be true. Similarly because of the constraint
always $(at(301) \oplus at\_room(303) \oplus \ldots \oplus at\_room(349) \oplus at\_elevator)$
one and only one of the fluents $at(301)$, $at(325)$, $at(327)$, $at\_elevator$, $at(other)$ is true. (To make it easier to analyze we will use the fluent at(other) and not distinguish the position of the robot at places other than at 301, 325, 327 and the elevator.) The notation used for the different states based on the possible combinations of these fluents is listed in the following table.

	$at(301)$	$at(325)$	$at(327)$	$at\_elevator$	$at(other)$
black_door_on_right	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,4}$	$s_{1,5}$
white_door_on_rt	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	$s_{2,4}$	$s_{2,5}$
wall_on_rt	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	$s_{3,4}$	$s_{3,5}$
corridor_on_rt	$s_{4,1}$	$s_{4,2}$	$s_{4,3}$	$s_{4,4}$	$s_{4,5}$

In the above table, $s_{i,j}$ denotes the state in which the fluent in the first column of row $i+1$ and the fluent in the $(j+1)^{th}$ column of the first row

[12]Please note that most results in this paper can be formulated in terms of states, state spaces and transition functions without talking about action theory. We felt that by doing that the connection between an action specification and the states and transition function it defines in a succinct and elaboration tolerant (often) way is pushed into the background, and we wanted to stress that states and transition functions are not often directly given but are rather specified by an action description. For that reason, we bring in action theories.

are true. For example, the fluents *wall_on_rt* and *at*(325) are true in the state $s_{3,2}$.

Since the action theory has two additional fluents, *just_started* and *app_corner*, to list all possible complete states of A we use the following notation:

- s_{i,j,app_corner} denotes the state where *app_corner* and *just_started* are true.
- $s_{i,j,\neg app\_corner}$ denotes the state where *just_started* is true and *app_corner* is false.
- s'_{i,j,app_corner} denotes the state where *app_corner* is true and *just_started* is false.
- $s'_{i,j,\neg app\_corner}$ denotes the state where *just_started* and *app_corner* are false.

We also use $s^*_{i,j,X}$ to denote that X can be either *app_corner* or $\neg$*app_corner* and $*$ can be either blank (meaning *just_started* is true) or $'$ (meaning *just_started* is false). We may also have one of them (the $*$ or the X) initialized. Also when we have $s^*_{i,j,X}$ as the member of a set, we are abusing notation, and what we mean is that the states obtained by initializing $*$ and X are member of that set. For example, when we say $S = \{s^*_{i,j,X}\}$ we mean $S = \{s_{i,j,app\_corner}, s_{i,j,\neg app\_corner}, s'_{i,j,app\_corner}, s'_{i,j,\neg app\_corner}\}$.

So far we have simply described all possible combinations of the fluents of the action theory.

Because of the domain constraints of A, not all fluent combinations are states. We list the fluent combinations which are not states below:

- Due to the domain constraint

  $$\textbf{always} \quad (corridor\_on\_rt \Rightarrow \neg at(other))$$

  $s^*_{4,5,X}$ are not states in A.
- Due to

  $$\textbf{always} \quad (app\_corner \Rightarrow \neg at(other))$$

  the states $s^*_{i,5,app\_corner}$ are not states in A.
- Due to

  $$\textbf{always} \quad (corridor\_on\_rt \Rightarrow \neg app\_corner)$$

  the states s^*_{4,j,app_corner} are not states in A.

We can now list the set S of all possible states the robot may be in as follows:

$$\begin{array}{lll} S = & \{s^*_{i,j,X} & \text{for } i = 1,\ldots,3; j = 1,\ldots,4\} \bigcup \\ & \{s^*_{i,5,\neg app\_corner} & \text{for } i = 1,\ldots,3\} \bigcup \\ & \{s^*_{4,j,\neg app\_corner} & \text{for } j = 1,\ldots,4\} \end{array}$$

In the following table we show the effect of various actions on some of the states of the robot.

Action (a)	State (s)	The state corresponding to $Res(a, s)$
$turn\_right\_90$	$s^*_{4,j,X}$	$s^*_{3,j,X}$
$turn\_180$	$s^*_{2,j,X}$	$s^*_{1,j,X}$
$detele\_started$	$s^*_{3,j,X}$	$s'_{3,j,X}$
$go\_forward$	$s^*_{1,5,\neg app\_corner}$	$s^*_{3,5,\neg app\_corner}$
	$s^*_{1,j,X}$ for $(j = 1, \ldots, 4)$	s^*_{3,j,app_corner}
	s^*_{3,j,app_corner} for $(j = 1, \ldots, 4)$	$s^*_{4,j,\neg app\_corner}$
	$s'_{3,1,\neg app\_corner}$	$s'_{1,5,\neg app\_corner}$
	$s'_{3,2,\neg app\_corner}$	$s'_{1,3,\neg app\_corner}$
	$s'_{3,3,\neg app\_corner}$	$s'_{1,5,\neg app\_corner}$
	$s'_{3,4,\neg app\_corner}$	$s'_{1,1,\neg app\_corner}$
	$s'_{3,5,\neg app\_corner}$	$s'_{1,Y,\neg app\_corner}(Y \in \{2, 4, 5\})$

3 Formal Characterization of Simple Control Modules

In our characterization we do not expect our control module to necessarily be a universal plan. (The correctness of a universal plan is given in [Sch87].) Thus we need to characterize what it means for a control module to *behave correctly* when it is executed in a certain (important and/or critical and/or most plausible) set of states S. A simple characterization in the *absence of exogenous actions*, and when the goal is to achieve a fluent formula G is quite straight forward and can be intuitively described as follows: We say a control module M behaves correctly with respect to a particular state s (from S), if when M is executed in s, M successfully terminates in a state where G is true. We say M behaves correctly with respect to S, if M behaves correctly with respect to all states in the set S.

But now we need to take into account exogenous actions, the main component of a dynamic environment. We propose to account them in a different manner than done in [KBSD97, DKKN95], where exogenous actions are combined with the agents actions thus resulting in non-deterministic (or probabilistic) effects of actions. In our approach the notion of correctness of a control module with respect to achieving a goal from a particular state s is as described in the previous paragraph and assumes no interference by exogenous actions. We take into account exogenous actions by expanding the set of states S to the set of all states that the agent may reach while executing M and due to exogenous actions. This larger set of states is referred to as the closure[13] of S with respect to the module M and the action theory A (which includes the theory for the exogenous actions), and is denoted by $Closure(S, M, A)$. Thus in presence of exogenous actions, we say M behaves correctly with respect to S, if M behaves correctly with respect to all states in the set $Closure(S, M, A)$. We now define the closure and the correctness of achievement modules more formally.

Definition 3.1 Let S be a set of states, M be a control module, and A be an action theory. We say a set of states S' is a closure of S w.r.t. M and A, if S' is a minimal (w.r.t. subset ordering) set of states that satisfies the following conditions:

[13]A similar notion is sometimes used in defining the final state reached due to ramification constraints.

(i) $S \subseteq S'$.

(ii) If $s \in S'$ and a is an action in A that can occur independent of the robot in the state s, then $Res(a, s) \in S'$[14] . (iii) If $s \in S'$ and there exist a simple control rule in M whose LHS is satisfied by s then $[RHS]s \in S'$[15] .
□

Proposition 3.1 For any set of states S, control module M, and action theory A, there is a unique set of states S' which is the closure of S w.r.t. M and A.

Proof: In Appendix A. □

We refer the closure of S w.r.t. M and A by $Closure(S, M, A)$. Also by $Closure(S, A, A)$, we denote the set of all states that can be reached from S by executing any sequence of actions (both exogenous and the actions doable by the robot) from a state in S.

Definition 3.2 A set of states S is said to be *closed* w.r.t. a control module M and an action theory A if $S = Closure(S, M, A)$. □

In the following example we illustrate the computation of Closure with respect to the control module $Go\_Clockwise\_1room$ and several theories of exogenous actions.

Example 3.1 Consider our example about the robot in an office floor. When describing the control module that takes the robot to the next room in the clockwise direction we assumed that in the initial state the robot will have $just\_started$ true and it will have the black door on its right.

Based on these assumptions the set of initial states for the robot, which we will denote by S_1, in the notation described in Section 2.5 are:

$S_1 = \{s_{1,1,X}, s_{1,2,X}, s_{1,3,X}, s_{1,4,X}, s_{1,5,\neg app\_corner}\}$

We will now highlight several closures of S_1 with respect to the control module $Go\_Clockwise\_1room$ and several theories of exogenous actions.

Following are some exogenous actions that we consider in computing the closure.

- $hand\_turn\_180$ - This is an action that represents the mischief of a passer-by who turns the robot 180 degrees when it has black door at its right. This action changes the orientation of the robot such that it has white door on its right. I.e, it changes the state of the robot from $s^*_{1,j,X}$ to $s^*_{2,j,X}$.
- $put\_on\_corridor$ - This action, again doable by a passer-by, puts the robot such that it has the corridor on its right, when it was initially approaching the corner or was just past the corner. I.e, it changes the state of the robot from s^*_{3,j,app_corner} to $s^*_{4,j,\neg app\_corner}$, or from $s^*_{3,j,\neg app\_corner}$ to $s^*_{4,j,\neg app\_corner}$.

[14] In this section by $Res(a, s)$ we denote the state corresponding to the situation $Res(a, s)$. Formally, this state is expressed by the set $\{f : holds(f, Res(a, s))$ is entailed by the theory$\}$.

[15] Recall that by $[a_1, \ldots, a_n]s$ we denote the state reached after executing the sequence of actions $a_1, \ldots, a_n$ from the state s.

- *corner_change*: This action, again doable by a passer-by, either advances a robot approaching a corner past the corner or vice versa. I.e., it changes the state of the robot from s_{i,j,app_corner} to the state $s_{i,j,\neg app\_corner}$ $(i = 1, \ldots, 4,\ j = 1, \ldots, 4)$, or from $s_{i,j,\neg app\_corner}$ to s_{i,j,app_corner}.

Let us consider four action theories:
A_1: It has no exogenous actions;
A_2: It has only the exogenous action *hand_turn_180*;
A_3: It has the exogenous actions *hand_turn_180* and *put_on_corridor*; and
A_4: It has all the exogenous actions.

We will now illustrate the computation of $Closure(S_1, M, A_1)$ and list the values of the other closures. Their detailed computation is given in the Appendix B.

Computing $Closure(S_1, M, A_1)$
We compute the closure by computing the least fixpoint of T_{M,A_1,S_1}.
Since there are no exogenous actions in A_1, $R_{A_1,S_1}(X) = \emptyset$ for every set X.
Therefore, $T_{M,A_1,S_1}(X) = S_1 \cup X \cup R_{M,S_1}(X)$.

1. Since $R_{M,S_1}(\emptyset) = \emptyset$, $T^0_{M,A_1,S_1}(\emptyset) = S_1$.
2. Since in each state in S_1 the only rule that can be applied is the rule **if** $just\_started, black\_door\_on\_rt$ **then** $go\_forward$; hence,

$$\begin{aligned} T^1_{M,A_1,S_1}(\emptyset) &= T_{M,A_1,S_1}(S_1) \\ &= S_1 \cup \{s_{3,1,app\_corner},\ s_{3,2,app\_corner}, \\ &\quad s_{3,3,app\_corner},\ s_{3,4,app\_corner},\ s_{3,5,\neg app\_corner}\} \\ &= X_1 \end{aligned}$$

3. In the states in $X_1 \setminus S_1$, the rule **if** $just\_started, wall\_on\_rt$ **then** $delete\_started$ is the only rule applicable and hence $just\_started$ becomes $false$ when this rule is executed in these states. Hence,

$$\begin{aligned} T^2_{M,A_1,S_1}(\emptyset) &= T_{M,A_1,S_1}(X_1) \\ &= X_1 \cup \{s'_{3,1,app\_corner},\ s'_{3,2,app\_corner}, \\ &\quad s'_{3,3,app\_corner},\ s'_{3,4,app\_corner},\ s'_{3,5,\neg app\_corner}\} \\ &= X_2 \end{aligned}$$

4. Now, the rule **if** $\neg just\_started, wall\_on\_rt$ **then** $go\_forward$ is the only rule that can be applied to the states in $X_2 \setminus X_1$. Therefore,

$$\begin{aligned} T^3_{M,A_1,S_1}(\emptyset) &= T_{M,A_1,S_1}(X_2) \\ &= X_2 \cup \{s'_{4,1,\neg app\_corner},\ s'_{4,2,\neg app\_corner}, \\ &\quad s'_{4,3,\neg app\_corner},\ s'_{4,4,\neg app\_corner},\ s'_{1,2,\neg app\_corner}, \\ &\quad s'_{1,4,\neg app\_corner},\ s'_{1,5,\neg app\_corner}\} \\ &= X_3 \end{aligned}$$

5. In this step, the rule **if** $corridor\_on\_rt$ **then** $turn\_rt\_90$ is the only one applicable in the states $s'_{4,j,\neg app\_corner}$ and the rule **if** $\neg just\_started, black\_door\_on\_rt$ **then** $HALT$ is the only one applicable in the states $s'_{1,j,\neg app\_corner}$. Therefore,

$$\begin{aligned} T^4_{M,A_1,S_1}(\emptyset) &= T_{M,A_1,S_1}(X_3) \\ &= X_3 \cup \{s'_{3,1,\neg app\_corner},\ s'_{3,2,\neg app\_corner}, \\ &\quad s'_{3,3,\neg app\_corner},\ s'_{3,4,\neg app\_corner}\} \\ &= X_4 \end{aligned}$$

6. For the states in $X_4 \setminus X_3$, the rule **if** $\neg just\_started, wall\_on\_rt$ **then** $go\_forward$ is the only rule that is applicable. Since, one of the resulting state ($s'_{1,5,\neg app\_corner}$) is already in X_3; this leads to

$$\begin{aligned} T^5_{M,A_1,S_1}(\emptyset) &= T_{M,A_1,S_1}(X_4) \\ &= X_4 \cup \{s'_{1,3\neg app\_corner}, \ s'_{1,1\neg app\_corner}\} \\ &= X_5 \end{aligned}$$

7. Since, $T_{M,A_1,S_1}(X_5) = X_5$,

$$\begin{aligned} Closure(S_1, M, A_1) = \ & \{s_{1,j,X} \ | j = 1, \ldots, 4\} \cup \\ & \{s'_1, j, \neg app\_corner \ | j = 1, \ldots, 4\} \cup \\ & \{s_{1,5,\neg app\_corner}, \ s'_{1,5,\neg app\_corner}\} \cup \\ & \{s_{3,j,app\_corner} \ | j = 1, \ldots, 4\} \cup \\ & \{s'_{3,1,X} \ | j = 1, \ldots, 4\} \cup \\ & \{s_{3,5,\neg app\_corner}, \ s'_{3,5,\neg app\_corner}\} \cup \\ & \{s'_{4,j,\neg app\_corner} \ | j = 1, \ldots, 4\} \end{aligned}$$

The other closures are as follows:

$$\begin{aligned} Closure(S_1, M, A_2) = \ & Closure(S_1, M, A_1) \cup \\ & \{s_{2,j,X} \ | j = 1, \ldots, 4\} \cup \{s_{2,5,\neg app\_corner}\} \cup \\ & \{s'_{2,j,\neg app\_corner} \ | j = 1, \ldots, 5\} \end{aligned}$$

$$\begin{aligned} Closure(S_1, M, A_3) = \ & Closure(S_1, M, A_2) \cup \\ & \{s_{3,j,\neg app\_corner} \ | j = 1, \ldots, 4\} \cup \\ & \{s_{4,j,\neg app\_corner} \ | j = 1, \ldots, 4\} \end{aligned}$$

$$Closure(S_1, M, A_4) = Closure(S_1, M, A_3)$$

□

3.1 Correctness of Achievement Control modules

Now that we have characterized the closure of a set of initial states w.r.t. a control module and an action theory, to prove correctness we will have to show that for each state s in the closure, if the control module is executed starting from s, then *in the absence of any exogenous actions* the control module will terminate in a state where the goal is satisfied. The reason we can get away with the assumption of no exogenous actions is because they are considered when computing the closure. So if during the execution of the control module an exogenous action does happen, then the robot will get distracted from its current execution, but it will reach a state in the closure, from which the control module can take it to the goal. Of course if the robot is continuously harassed by exogenous actions it will not reach the goal; but if there is a *window of non-interference* where there are no exogenous actions then it will reach the goal. (Although this is a drawback for a robot that is being continuously harassed, there is no easy way out. One approach would be to avoid getting into such a situation altogether, and another would be to be able to recognize it as a *failure* and trigger a recovery routine. The first approach is taken in [KBSD97, DKKN95]. But their drawback is that the robot becomes too conservative and avoids too many situations and may consider certain goals unachievable which will be considered achievable – albeit requiring a window of non-interference – in our framework. There are only some preliminary work [TSG95, TS95, San97] done on the second approach.)

To complete our formalization we now define the unfolding of a control module with respect to a state. Intuitively, given a state s and a simple control module M, the unfolding of M w.r.t. s is the sequence (possibly infinite) of actions that the control module will execute starting from s, in the absence of any exogenous actions.

In the following we say that a rule **if** LHS **then** RHS is applicable in a state s if LHS is satisfied in s and $[RHS]s$ is defined.

Definition 3.3 For an achievement (or a mixed) control module M, $\mathcal{U}_M$ the unfolding function of M from states to sequences of actions is defined as follows:

(i) For a state s, if all rules applicable in s, have RHS as HALT then $\mathcal{U}_M(s) = []$.

(ii) For a state s, if there exists no rule r in M which is applicable in s then $\mathcal{U}_M(s) = a_F^M$, where the action a_F^M is a special action in our action theory which denotes that the execution of M fails.

(iii) For a state s, if there is at least one rule applicable in s, then let α be the compound action that represents the concurrent execution of the RHS of all rules applicable in s and if $[\alpha]s$ is defined then $\mathcal{U}_M(s) = \alpha \circ \mathcal{U}_M([\alpha]s)$. □

We are now ready to define the correctness of an achievement control module with respect to a set of initial states, a control module and an action theory.

Definition 3.4 An achievement control module M is said to achieve goal G from a set of states S and w.r.t an action theory A (i.e., M is correct w.r.t. G, S and A), if for all s in $Closure(S, M, A)$, $\mathcal{U}_M(s)$ is finite and does not end with a_F^M and for all f in G, $\models holds(f, [\mathcal{U}_M(s)]s)$.

Furthermore, M is said to n-achieve goal G from S, if $max_{s \in S}|\mathcal{U}_M(s)| = n$. □

The notion of n-achievement is significant from the point of view of computation, where we might want our control module to be such that in the absence of exogenous actions it reaches the goal in less than n steps. In that case the number n represents the length of the *window of non-interference* that is necessary for the robot to achieve its goal.

Also we can easily generalize the above definition to the case where the goal G is a formula instead of a set of fluent literals.

3.2 Correctness of the *Go_clockwise_1room* module

Continuing with our running example we would like to show that the control module *Go_clockwise_1room* achieves the goal $\{\neg just\_started, black\_door\_on\_rt\}$ from the set of all possible states, S.

By Definition 3.4, we need to show that for all $s \in S$, $\mathcal{U}_M(s)$ is finite and does not end with a_M^F and for all f in the goal G, $\models holds(f, [\mathcal{U}_M(s)]s)$.

It is easy to see that for any state s in S, there is only one rule in the control module *Go_clockwise_1room* which is applicable in that state. This guarantees that the unfolding function unfolds to sequences of simple actions (no compound actions) for all states in S, and also that $\mathcal{U}_M(s)$ does not end with a_F^M for any s in S.

In Appendix B we compute $\mathcal{U}_M(s)$ for all possible states s, and show that the goal is true in each of those states.

3.3 Correctness of maintainance control modules

So far we have discussed the correctness of achievement control modules. Recall that maintainance control modules are supposed to maintain a goal. In the absence of exogenous actions it means that the robot should not do any action that might make the maintainance goal false. But in the presence of exogenous actions the robot with no control over those actions can only strive to make the maintainance goal true if they are made false by some exogenous actions. In other words a maintainance control module can not guarantee that the maintainance goal will never be false; *it can only guarantee that the robot won't make it false by its own actions and if it is made false by exogenous actions, then the robot will act towards making it true, and given a sufficient window of non-interference from exogenous actions, the robot will make the maintainance goal true.* (Our notion of maintainance is weaker than the notion of stability in discrete event dynamic systems [OWA91, RW87a, RW87b]. We discuss this difference later in Section 3.4.) We now formalize this notion of correctness of maintainance control modules.

Definition 3.5 For a maintainance control module M, $\mathcal{U}_M$ the unfolding function of M from states to sequences of actions is defined as follows:

(i) For a state s, if all rules applicable in s have RHS as SUSPEND then $\mathcal{U}_M(s) = []$.

(ii) For a state s, if there exists no rule r in M which is applicable in s then $\mathcal{U}_M(s) = a_F^M$, where the action a_F^M is a special action in our action theory which denotes that the execution of M fails.

(iii) For a state s, if there are several rules applicable in s, then $\mathcal{U}_M(s) = \alpha \circ \mathcal{U}_M([\alpha]s)$, where α is the compound action representing the concurrent execution of the RHS of all rules applicable in s. □

Definition 3.6 A maintainance control module M is said to maintain a goal G from a set of states S and w.r.t an action theory A, if for all s in $Closure(S, M, A)$, $\mathcal{U}_M(s)$ is finite and does not end with a_F^M and for all f in G, $\models holds(f, [\mathcal{U}_M(s)]s)$.

Furthermore, M is said to n-maintain goal G from S, if $max_{s \in S}|\mathcal{U}_M(s)| = n$. □

Intuitively when we say that a control module M n-maintains a goal G it means that at any time if the robot is not in a state where G is satisfied, then it will get back to a state where G is satisfied within n steps, if there is no interference in between.

Consider the maintainance control module $Maintain\_Siren$. It can be easily shown that it maintains the goal $\neg off\_siren$ from the set of initial states $\{\emptyset\}$ in presence of an exogenous action that may suddenly make off_siren true.

Correctness of mixed control module can be defined in a similar manner and we can show that the mixed control module obtained by adding the control rule

if off_siren **then** $turn\_on\_siren$, to the control module $Go\_clockwise\_1room$ will not only achieve the goal of going to the next room in the clockwise direction, but will also maintain the goal $\neg off\_siren$.

3.4 Other formulations of maintainance

In the previous section we described our formulation of maintaining a goal, and correctness of control modules with respect to our definition.

But in some cases our definition can be considered weak. For example, in one of the stronger formulations the agent is not allowed to get to an unacceptable state, regardless of what exogenous actions happen. This is used in policy determination algorithms for MDP (Markov decision process) based approaches [DKKN95, KBSD97]. (We compare our work to these works in additional detail in Section 4.1.) Another formulation that is stronger than our notion, and used in the discrete event dynamic system literature [OWA91, RW87a, RW87b] is the notion of 'stability' where an acceptable state must be reached within a finite number of transitions and then be visited infinitely often. The difference here is that it requires that an acceptable state be reached in a finite number of transitions, regardless of the presence of an window of non-interference. Thus modules that satisfy the maintainability criteria may not guarantee stability.

We feel that the particular stronger formulations mentioned above may be too strong and often there may not be any control module satisfying such a criterion. Moreover, in domains where the robots or agents actions are deterministic (such as in case softbots, and database updates), our approach of not combining the agent's action with the environments action, and use of the maintainance criteria seem to be more appropriate. Nevertheless, we now briefly discuss some possible in between formulations.

Let G be the set of goals states and by $S(X, exo, m)$ we denote the set of states that can be reached by any arbitrary sequence of m exogenous actions from any state in the set X.

For a number m, we define $core(G, m)$ to be the largest set of states satisfying the following properties.

- $core(G, m) \subseteq G$
- $S(core(G, m), exo, m) \subseteq G$
- $S(core(G, m), exo, m) \setminus G \neq \emptyset$
- There exists a robots action that can take the robot from any state in $S(core(G, m), exo, m)$ to a state in $core(G, m)$.

Now, we can define correctness by requiring that when the agent is ready to act (using its control module M) the state of the world is among the states in $S(core(G, m), exo, m)$. The number m can be varied by varying the cycle time of executing the control module M. The faster the cycle, the lesser will be the number of exogenous actions that can happen in between successive robots actions, the lesser the value of m, and the closer will be $core(G, m)$ to G. We plan to study further on this, particularly on its impact on mixed control modules, in the sequel.

4 Sufficiency conditions for correctness of simple control modules

4.1 Sufficiency conditions for achievement control modules

The definitions of the previous section formalize the notion of correctness of a control module as a whole. Our goal in this section is to explore sufficiency conditions that will (sometimes) allow us to verify whether a control module achieves a goal or not without actually constructing its unfolding function and explicitly verifying the conditions in Definition 3.4. Moreover we are also interested in the notion of the *correctness of an individual control rule.* This is important in the development of a control module because control rules are often written one by one by an expert, or are learned individually, or even if developed automatically, they are often incrementally added to an existing module; and we need to have a way to evaluate the correctness of such an individual rule regardless of what is in the rest of the module.

Thus in contrast to the approach in [KBSD97, DKKN95] we will pay special attention to sufficiency conditions for correctness of individual rules. We will refer to such rules as *sound.* Our goal is to show that a control module which is a *complete* (w.r.t. a set of states S) collection of such sound rules guarantee that the module will achieve its goal from S, where *completeness w.r.t.* S is defined as the module having at least one applicable rule for each state in S.

Inspired by Kaelbling and Rosenschein we say that a simple control rule r is intuitively sound if for any state where r is applicable, the *action* in the RHS of r '*leads to the goal.*'

We formally define this intuitive notion as follows: An action *a leads to a goal* from a state s if there exists a plan with minimal cost from s which achieves the goal and has a as its first action, where actions have an associated cost (a positive integer) and the cost of a plan is the sum of the cost of each of the actions in that plan.

Before formally defining the soundness condition we would like to point out that weaker definitions of the notion of an action leading to a goal, such as the action is the first action of any plan or even any minimal plan, are not sufficient. The following control module to go to the elevator illustrates our point.

Module : Goto_elevator_3

if $at\_room(X), X \geq 325$ **then** $Go\_clockwise\_1room$
if $at\_room(323)$, **then** $Go\_anticlockwise\_1room$
if $at\_room(321)$, **then** $Go\_clockwise\_1room$
if $at\_room(X), X \leq 319$ **then** $Go\_anticlockwise\_1room$
if $at\_elevator$ **then** $HALT$ □

The above module is complete in the sense that it has control rules for each possible state. Also for each possible state there is a unique rule in the above module that is applicable to that state, and the RHS of that rule is an action that is the first action of a minimal plan to the goal. To verify that let us consider the robot to be at room 323. From that state the robot has a minimal plan consisting of a sequence of $Go\_anticlockwise\_1room$ which takes the robot to the goal. Hence the action $Go\_anticlockwise\_1room$ is the first action of a minimal plan to the goal from the state $\{at\_room(323)\}$.

Similarly the action $Go\_clockwise\_1room$ is the first action of a minimal plan to the goal from the state $\{at\_room(321)\}$.

But if the robot is at room 323 or at room 321, and it executes the module $Goto\_elevator\_3$, it gets stuck in a loop.

It is also important that the cost of actions be positive. Otherwise control modules may also get into loops.

We now formally define the soundness condition based on the 'action leading to goal' notion. Later we will present an algorithms that automatically construct control modules using this notion.

Definition 4.1 *[Soundness]* (i)A simple control rule r is said to be *sound* w.r.t. a goal G and a set of states S (or w.r.t. (G, S)) if for all $s \in S$ such that r is applicable in s there exists a **minimal cost plan** that achieves G from s and has the RHS of r as its prefix.

(ii) A termination control rule r is said to be *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if its LHS satisfies G.

An achievement control module M is *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if each rule $r \in M$ is sound w.r.t (G, S). □

Note that in part (i) of the above definition, if there are several plans with the same minimal cost then any one of them can be used to satisfy the soundness criteria.

An important aspect of the above definition is the notion of 'minimal cost plan'. It raises the question of how exactly we assign cost to actions. A simple cost assignment is to assign each action a cost of 1. In that case a minimal cost plan corresponds to a shortest plan. One advantage of this is that many of the current planners (including most forward chaining planners) can easily find the shortest plan.

Besides the soundness and completeness condition, to make it easier to analyze and automatically construct control modules, we would like to avoid compound actions representing concurrent execution of actions. Moreover most planners do not construct plans that have such compound of actions. The following definition defines a condition that guarantees non-concurrent execution of actions.

Definition 4.2 An achievement control module M is said to be *sequential* w.r.t. a set of states S if for any $s \in S$, the RHS of all rules in M whose LHS is satisfied by s is the same. □

Observation 4.1 Consider a pair (M, S), where M is a sequential achievement control module w.r.t. a finite set of states S and S is closed w.r.t. M and a deterministic action theory. Then for any state s in S, the unfolding of M with respect to s does not contain any concurrent execution of actions. □

We are now ready to state our main theorem which states that completeness and soundness of an achievement control module M w.r.t a goal G guarantees that M achieves G.

Theorem 4.1 Let A be an action theory. Consider a pair (M, S), where S is a set of complete states, M is a simple control module sequential w.r.t. S, and S is closed w.r.t. M and A. Given a goal G, if M is sound w.r.t. G and S and complete w.r.t. S then M achieves G from S w.r.t. A.

Proof: In Appendix C. □

We will now consider the control modules in the previous sections and show their correctness by showing that they satisfy the soundness and the completeness criteria. Note that one advantage in using this approach of showing correctness is due to the fact that there now exist incremental planners [JB95, AIS88] which can determine the prefix of plans, without constructing the whole plan. This may be used to verify the soundness condition without actually constructing the minimal cost plans.

- The control module $Goto\_elevator\_1$ is sound and complete with respect to the set of states $\{\{at\_elevator\}, \{at(301)\}, \ldots, \{at(349)\}\}$, and the goal $at\_elevator$ when the action theory only consist of the action $Go\_clockwise\_1room$. We can then use Theorem 4.1 to verify its correctness.

  When we add the action $Go\_anticlockwise\_1room$, it is no longer sound, and we can no longer use Theorem 4.1 to verify its correctness.

- The control module $Goto\_elevator\_2$ is sound and complete with respect to the set of states $\{\{at\_elevator\}, \{at(301)\}, \ldots, \{at(349)\}\}$, and the goal $at\_elevator$ when the action theory consists of actions $Go\_clockwise\_1room$ and $Go\_anticlockwise\_1room$. Therefore, we can use Theorem 4.1 to verify its correctness.

- The control module $Go\_clockwise\_1room$ is sound and complete with respect to the set of all states $\{\{at\_elevator\}, \{at(301)\}, \ldots, \{at(349)\}\}$, and the goal $at\_elevator$, and the action theory in Section 2.3. Therefore, we can also use Theorem 4.1 to verify its correctness.

- The soundness and completeness conditions and Theorem 4.1 can be modified in an intuitive manner for mixed and maintainance control modules and we can use them to prove the correctness of the control module $Maintain\_Siren$.

4.2 Automatic construction of achievement control modules

In the last section we discussed how the soundness and completeness conditions can be used to verify the correctness of some control modules. *But its other important significance is that it can be used to automatically construct control modules.* In this section we give an algorithm to automatically construct achievement control modules. Given a set of states S, an action theory A and goal G the algorithm uses the soundness and completeness condition and Theorem 4.1 to construct a control module M that can reach the goal from any state in $Closure(S, M, A)$. (We assume here that G is achievable from all states in $Closure(S, A, A)$.) The main step of the algorithm is to add sound control rules to the control module for all states in $Closure(S, M, A)$. Normally to consider all states in the set $Closure(S, M, A)$ we need to computed the set first. But to compute that we need the control module M. We avoid this 'chicken-and-egg' problem by iteratively computing $Closure(S, M, A)$ and adding sound rules for states already in the current $Closure(S, M, A)$ until a fixpoint is reached where we have a control module M which has sound rules for all states in $Closure(S, M, A)$.

Algorithm 4.1

Input: Goal G, a set of states S, an action theory A for the robot and for the exogenous actions.

Output: A Control Module M that achieves G from $Closure(S, M, A)$.

Step 1 $M = \{\textbf{if } G \textbf{ then } HALT\}$,
$S_{in} = \{s' : s'$ is reachable from some state s in S by applying sequences of exogenous actions$\}$, and
$S_{out} = \emptyset$.

Step 2 While $S_{in} \neq S_{out}$
2.1 Pick a state s from $S_{in} \setminus S_{out}$
2.2 Find a minimal cost plan P from s that achieves G. Let a be the first action in P.
2.3 If P is a not a null plan then $M = M \cup \{\textbf{if } s \textbf{ then } a\}$;
2.4 $S_{in} = S_{in} \cup [a]s \cup \{s' : s'$ is reachable from $[a]s$ by applying sequences of exogenous actions$\}$ and $S_{out} = S_{out} \cup \{s\}$.

Step 3 Merge control rules in M which have the same RHS and whose LHS can be combined[16].

□

Proposition 4.1 Given a goal G, a set of states S and an action theory A if G is achievable from all states in $Closure(S, A, A)$, then Algorithm 4.1 generates a control module M such that M achieves G from $Closure(S, M, A)$.

Proof (sketch): The set $Closure(S, M, A)$ is iteratively computed in the step 2.4 and put in S_{in}. S_{out} is the set of states for which sound rules have been added to M in step 1 and step 2.3. Since step 2 terminates when $S_{out} = S_{in}$, after the algorithm terminates we have M to be sound and complete w.r.t. the goal G and the set of states $Closure(S, M, A)$. The algorithm is guaranteed to terminate because of our assumption that G is achievable from all states in $Closure(S, A, A)$. Hence by Theorem 4.1 this proposition is true. □

In the above proposition we require that G is achievable from all states in $Closure(S, A, A)$. This is a fairly stringent requirement. In the absence of this condition we can not guarantee that our algorithm will construct a control module M that achieves G from $Closure(S, M, A)$. In fact we can not guarantee that our algorithm will terminate. A slight modification of the algorithm, where we replace Step 2 by

> "While $S_{in} \neq S_{out}$ and there exists s in $S_{in} \setminus S_{out}$ from which there is a plan to reach the goal"

will guarantee termination, and if the algorithm terminates with $S_{in} = S_{out}$, then the control module M generated by the algorithm achieves G from $Closure(S, M, A)$. Also since the algorithm picks a minimal cost plan in Step 2.2, this can be used as a choice point to backtrack, to look for other ways to have $S_{in} = S_{out}$.

The above proposition guarantees the correctness of our automatic construction algorithm. But besides being correct, the control module generated by our algorithm is also 'optimal'; i.e., from any state the control module takes a minimal cost path to a goal state, when there are no exogenous actions. The control modules generated in [DKKN95, KBSD97] do not have this property.

4.3 Complexity of the automatic construction algorithm

It is clear that the complexity of the above algorithm depends on the size of the closure and the time it takes to find minimal cost plans. The worst case size of the closure is the total number of states, which is exponential in terms of the number of fluents. But we can limit the size of the closure by allowing a small number of exogenous activities in A and by starting with only few initial states. Similarly the complexity of finding a minimal cost program can be thought of as finding minimal cost paths in a graph, which using Dijkstra's algorithm is quadratic in the number of nodes in the graph. In general the number of nodes in our search graph could be exponential in terms of the total number of fluents. (The complexity of planning algorithms with respect to the number of fluents in most planning domains is listed in detail in [ENS95].) However, in reality, many of the fluent combinations (i.e. states) may violate the state constraints and thus may not be valid. Hence we believe that in some control modules and the corresponding environment the number of the nodes in the graph will not be too big, and will allow efficient searching of minimal cost paths. Nevertheless in view of the worst case exponential complexity, this algorithm and other algorithms in this paper should be used mostly off-line, when time is less of a concern.

We now present a different algorithm that uses the 'weakly leads' condition. In this algorithm, for every state in the closure we not only add a control rule satisfying the 'weakly leading' condition corresponding to that state, but also add several such rules corresponding to states that can be reached by the execution of just added control rules, until we reach a state which has been considered before. This prevents the control module from having an infinite sequence of action as the unfolding with respect to some state.

Algorithm 4.2

Input: Goal G, a set of states S, an action theory A for the robot and for the exogenous actions.

Output: A Control Module M that achieves G from $Closure(S, M, A)$.

Step 1 $M = \{\textbf{if } G \textbf{ then } HALT\}$,
$S_{in} = S \cup \{s' : s'$ is reachable from some state s in S by applying sequences of exogenous actions$\}$ and $S_{out} = \emptyset$.

Step 2 While $S_{in} \neq S_{out}$

2.1 Pick a state s from $S_{in} \setminus S_{out}$

2.2 If s satisfies G then $S_{out} = S_{out} \cup \{s\}$ and go to step 2. Otherwise, go to step 2.3

2.3 Find the minimal cost plan P from s that achieves G. Let the minimal plan be $a_1, \ldots, a_k$.

2.4 Let i be the smallest number such that $[a_1, \ldots, a_i]s \in S_{out}$. (If $[a_1, \ldots, a_i]s \notin S_{out}$ for $i = 1, \ldots, k$ then we set $i = k+1$.)

2.5 Let $M = M \cup \{\textbf{if } [a_1, \ldots, a_j]s \textbf{ then } a_{j+1} : 0 \leq j < i\}$;

2.6 $S_{in} = S_{in} \cup \{[a_1, \ldots, a_j]s \ : \ 0 \leq j < i\} \cup \{s' : s'$ is reachable from $[a_1, \ldots, a_j]s$ for some $j < i$ by applying sequences of exogenous actions$\}$ and
$S_{out} = S_{out} \cup \{[a_1, \ldots, a_j]s : 0 \leq j < i\}$.

Step 3 Merge control rules in M which have the same RHS and whose LHS can be combined.

□

Proposition 4.2 Given a goal G, a set of states S and an action theory A if G is achievable from all states in $Closure(S, A, A)$, then Algorithm 4.2 generates a control module M such that M achieves G from $Closure(S, M, A)$.
□

Proof: In Appendix C.

4.4 Automatic construction of maintainance and mixed control modules

The definition of soundness and completeness can be extended in an intuitive way to give us results similar to Theorem 4.1 that can be used to verify correctness of maintainance and mixed control modules. Moreover the two algorithms in the previous section can be appropriately modified to automatically construct maintainance and mixed control modules.

5 Sensing actions, conditional plans and their role in control modules

In the previous sections we developed the notion of correctness of a simple control module with respect to a goal, an action theory, initial states, and exogenous actions. We also gave sufficiency conditions for correctness of control rules. One of the assumptions that we had in the previous section was that after each sensing the robot has complete information about each fluent. Often the various sensing that needs to be done to satisfy the above assumptions may not be doable in all situations. For example, in the AAAI 96 robot contest the robot needs to know if a certain conference room is occupied or not. The sensing necessary to find this out can only be done if the robot is in or near the conference room. Moreover some of the sensing activities, such as analyzing an image or a pattern, may be so expensive (or time consuming) that we may only want to do it in certain states. For these reasons we would like to separate the sensing activities of a robot into two groups: *regular sensing* and *special sensing*. The sensing that the robot does in the sense and act cycle will be referred to as *regular sensing*, and the sensing that it does in the acting phase to determine values of certain fluents, such as finding if the conference room is occupied or not, will be referred to as *special sensing*. Special sensing actions will appear as an action in the 'then' part of a control rule and as a sensing action in a conditional plan [Lev96].

In the next few sections of this paper we consider control modules with special sensing actions and formulate their correctness with respect to action theories that allow such actions [Moo85, SL93, LTM97]. (A plan based on such a theory could be a conditional plan [Lev96] and may achieve knowledge goals [GEW96, GW96].) We also present sufficiency conditions for correctness of individual control rules and show how it can be used to construct control modules with sensing actions.

Special sensing actions also allow us to construct conditional plans from *incomplete* states – where the robot does not have complete knowledge about the world, for which simple plans may not exist.

5.1 Control modules with sensing actions

We now extend the simple control modules of section 2 with two new features: they may have explicit sensing actions in the 'then' part of the rules and may have fluent expressions of the form $u(f)$ – meaning the truth-value of f is unknown, in the 'if' part. Besides these changes, the operational semantics of such control modules remains unchanged. We now give an example of such a control module.

Consider a robot which can perform the actions: *check_door_lock, flip_lock, and push_door*. Intuitively if the robot performs the action *check_door_lock*, it will know if the door is locked or not. If it performs $flip\_lock$ then the door becomes unlocked if it is locked, and becomes locked if it is unlocked. If it performs *push_door* when the door is unlocked the door opens. Let us now consider a control module which a mobile robot can execute to open the door.

Example 5.1 Control Module – *Open_Door*
if $\neg door\_open$, $u(door\_locked)$ **then** $check\_door\_lock$
if $\neg door\_open$, $door\_locked$ **then** $flip\_lock$
if $\neg door\_open$, $\neg door\_locked$ **then** $push\_door$
if $door\_open$ **then** $HALT$ □

Our goal now is to extend our formulation of correctness of simple control modules to control modules that have sensing actions. To do that the corresponding theory of action, must allow formalization of sensing actions.

5.2 Action theory with sensing actions

For our purpose the main enhancement we need in our action theory, with respect to the action theory in Section 2.4, is that the theory allow sensing actions – actions that can change the state of the robot's knowledge.

As before, our action theory will have two kinds of actions: one that the robot can perform (both simple and sensing actions), and the other (also both simple and sensing actions) that may happen independent of the robot and which is beyond the control of the robot. Intuitively exogenous sensing actions corresponds to an outside agent telling the robot - this includes a human agent entering such data through a keyboard, the truth value of some fluents, of whose truth value the robot did not have any prior knowledge. For non-sensing actions we have effect axioms that describe the effect of the actions and for sensing actions we have axioms that describe the knowledge that may be gained by executing that action. As an example, the action theory of the robot for the Example 5.1 can be described by the following axioms.

Causal Rules describing actions in module *Open_Door*
$check\_door\_lock$ **determines** $door\_locked$
$push\_door$ **causes** $door\_open$ **if** $\neg door\_locked$
$flip\_lock$ **causes** $door\_locked$ **if** $\neg door\_locked$
$flip\_lock$ **causes** $\neg door\_locked$ **if** $door\_locked$

In the above rules, *check_door_lock* is a sensing action while the other two are non-sensing actions.

Recall that we use action theories to define an entailment relation between specification of actions and queries of the form *goal* **after** *plan*. We then use this entailment relation to formulate correctness of a control module with respect to a set of initial states, by unfolding the control module with respect to each of the initial state and checking if the plan obtained by unfolding does indeed achieves the goal. For simple control modules, the unfolding produced simple plans, which were sequences of actions.

With sensing actions we can extend our formulation of correctness with respect to *incomplete* states – which we will denote by a pair $\langle T, F \rangle$, where T and F are the set of fluents which the robot knows has truth value $true$, and $false$ respectively. But unfolding a control module with respect to an incomplete state may not produce a simple plan consisting of sequences of actions. For example, unfolding the control module *Open_Door*, with respect to the incomplete state $(\langle\emptyset, \{door\_open\}\rangle)$, where the robot is unaware of the truth value of any fluents besides *door_open*, intuitively results in the following plan.

Conditional Plan - *Plan_Open_Door*
check_door_lock;
Case
  $\neg door\_locked \rightarrow push\_door$;
  $door\_locked \rightarrow flip\_lock, push\_door$;
Endcase

We refer to such a plan as a conditional plan. Note that like plans consisting of sequences of actions, conditional plans will take an agent to its goal from a particular state assuming there are no exogenous actions. In contrast, the control module will take an agent to its goal from among a set of states and in presence of a set of anticipated exogenous actions. We now formally define a conditional plan.

Definition 5.1 *[Conditional Plan]*

- The empty plan [] is a conditional plan.
- If a is an action then a is a conditional plan.
- If c_1 and c_2 are conditional plans, then $c_1; c_2$ is a conditional plan.
- If $c_1, \ldots, c_n$ are conditional plans such that at least one of the c_i's is not empty, and p_{ij}'s are fluents then the following is a conditional plan. (Such a plan is referred to as a *case plan*).

  Case
  $p_{1,1}, \ldots, p_{1,m_1} \rightarrow c_1$
  $\vdots$
  $p_{n,1}, \ldots, p_{n,m_n} \rightarrow c_n$
  Endcase

  where $p_{1,1}, \ldots, p_{1,m_1}, \ldots,\ p_{n,1}, \ldots, p_{n,m_n}$ are mutual exclusive (but not necessary exhaustive).

  In the above, $p_{i,1}, \ldots, p_{i,m_i} \rightarrow c_i$ will be referred to as a case statement of the plan C.
- Nothing else is a conditional plan. $\square$

Note that any non-empty conditional plan C can be represented as a sequence of non-empty plans $C_1; C_2; \ldots; C_k$ where $k \geq 1$ and C_i is a sequence of actions or a case plan.

Based on the above discussion the enhanced action theory that we need should allow states to encode what the robot knows, and allow reasoning about sensing actions and conditional plans.

Some of the theories of the above kind are described in [Moo85, Haa86, SL93, LTM97, BS97]. In [Moo85, SL93], states are Kripke models, while in [LTM97] states are sets of 3-valued interpretations. To make matters simple we follow the approximation approach in [BS97], where a state is a pair $\langle T, F\rangle$, where T (resp. F) contains the fluents which have the truth value *true* (resp. *false*). The theories in [Moo85, SL93] and [LTM97] can be easily adapted to reason with such states. Once a state is defined, the next step involves defining a transition function Φ that encodes the effect on an action on a state. For a sensing action a that determines a fluent f, whose value is not known in a state $\langle T, F \rangle$, $\Phi(a,s)$ is the set of states $\{\langle T \cup \{f\}, F\rangle, \langle T, F \cup \{f\}\rangle\}$. Since $\Phi(a,s)$ could be a set of states, we say $\models holds(f,[a]s)$ is true, if f is true (or false) in every state in $\Phi(a,s)$. This is then further generalized in an intuitive way to define when $\models holds(f,[plan]s)$ is true, where *plan* is a conditional plan.

Using an action theory that encodes the above ideas, it can be shown that $\models holds(door\_open, [P]\langle\emptyset,\emptyset\rangle)$ where P stands for the conditional plan *Plan_Open_Door* specified earlier.

We do not intend to advocate any particular theory of action for the rest of the formulation. *Any theory that formalizes sensing actions by defining Φ and the above discussed entailment relation, and that allows states that encode the knowledge of the robot about the world* is suitable for the rest of the formulation.

6 Formal Characterization of Control Modules with sensing actions

As before, to take into account exogenous actions we first define the closure of a set of states with respect to a control module and an action theory. This definition is very similar to Definition 3.1. The only difference is that we assume that the robot may be told by an outside agent about truth values of fluents that it does not know. We do not require this assumption to be explicitly stated as part of the action theory.

Definition 6.1 Let S be a set of states, M be a control module, and A be an action theory with a transition function Φ. By $Closure(S,M,A)$ we denote the smallest set of states that satisfy the following conditions:

- $S \subseteq Closure(S,M,A)$.
- For any state σ in S, if σ' is an extension[17] of σ then σ' is in $Closure(S,M,A)$.
- If $s \in Closure(S,M,A)$ and a is an action in A that can occur independent of the robot then $\Phi(a,s) \subseteq Closure(S,M,A)$.
- If $s \in Closure(S,M,A)$ and there exist a rule in M whose LHS is satisfied by s, then $\Phi(RHS,s) \subseteq Closure(S,M,A)$. □

Proposition 6.1 For any set of states S, control module M and action theory A, there is a unique set of states S' which is the closure of S w.r.t. M and A.

Proof: Similar to the proof of Proposition 3.1. □

We refer to the closure of S w.r.t. M and A by $Closure(S,M,A)$.

Definition 6.2 A set of states S is said to be *closed* w.r.t. a control module M and an action theory A if $S = Closure(S,M,A)$. □

[17] A state $\langle T', F' \rangle$ is said to be an extension of a state $\langle T, F \rangle$ iff $T \subseteq T'$ and $F \subseteq F'$.

We will now formally characterize the effect of executing control modules with special sensing actions. Intuitively a control module M executed in a state s executes a conditional plan. If s is complete then the conditional plan that is executed is *a sequence of actions – which we refer to as a simple or a linear plan.* When s is incomplete, the conditional plan that is executed may not be a sequence of actions, and may include sensing actions and case statements.

We now formally define the unfolding function of a control module. As before, we say that a control rule **if** LHS **then** RHS is *applicable* in s if LHS is satisfied in s and $[RHS]s$ is defined.

Definition 6.3 For an achievement (or a mixed) control module M, we say $\mathcal{U}_M$ is an *unfold model* if the following conditions are satisfied.

- For a state s if all rules applicable in s have RHS as HALT then $\mathcal{U}_M(s) = []$.
- For a state s, if there exists no rule r in M which is applicable in s then $\mathcal{U}_M(s) = a_F^M$, where the action a_F^M is a special action in our action theory which denotes that the execution of M fails.
- For a state s, if there is at least one rule applicable in s, then let α be the compound action that represents the concurrent execution of the RHS of all rules applicable in s.
  - If $\Phi(\alpha, s) = \{s_1, \ldots, s_n\}$ with $n > 1$, then

    $$\begin{array}{ll} \mathcal{U}_M(s) = \alpha; & \\ & \text{Case} \\ & s_1 \rightarrow \mathcal{U}_M(s_1) \\ & \vdots \\ & s_n \rightarrow \mathcal{U}_M(s_n) \\ & \text{Endcase} \end{array}$$

  - else, if $\Phi(\alpha, s) = \{s'\}$, then

    $\mathcal{U}_M(s) = \alpha \circ \mathcal{U}_M(s')$. □

Definition 6.4 An achievement control module M with sensing actions is said to achieve goal G from a set of states S and w.r.t. an action theory A (i.e., M is correct w.r.t. G, S and A), if for all s in $Closure(S, M, A)$, $\mathcal{U}_M(s)$ is finite and does not end with a_F^M and for all f in G, $\models holds(f, [\mathcal{U}_M(s)]s)$. □

7 Sufficiency conditions and automatic construction of control modules with sensing actions

Our motivation here is similar to that of in Section 4.1; we would like to obtain sufficiency conditions for the correctness of control modules with sensing actions. This will be useful in verification of the correctness of control modules, and also for automatic construction of control modules that use sensing actions to counter incompleteness.

As in Section 4.1 the sufficiency condition here will also have two parts: soundness and completeness conditions. The completeness condition here will be similar to the one in Section 4.1. Although the intuitive idea behind the soundness condition here will be the same as before in Section 4.1, one difference is that unfolding of control modules with respect to incomplete states may now give us a conditional plan, not just a simple plan. *Because of this we need to extend our earlier definition of a minimal plan and a minimal cost plan to conditional plans.* We start with definition of a sub-plan.

Definition 7.1 *[Sub-plans of conditional plans]* Given a conditional plan C. A plan C' which is not identical to C is called a sub-plan of C if

1. C' is the empty plan [], or
2. C is a simple plan $a_1; \ldots; a_n$ and $C' = a_{j_1}; \ldots; a_{j_k}$ where $j_1, \ldots, j_k$ is a nonempty subsequence of $1, \ldots, n$ $(1 \leq j_1 < \ldots < j_k \leq n)$, or
3. C is a case plan

   $$\begin{array}{l} \mathrm{C} = \mathrm{Case} \\ \quad\quad p_{1,1}, \ldots, p_{1,m_1} \rightarrow c_1 \\ \quad\quad \vdots \\ \quad\quad p_{n,1}, \ldots, p_{n,m_n} \rightarrow c_n \\ \quad \mathrm{Endcase} \end{array}$$

   and

   (a) either $C' = c'_i$ for some $1 \leq i \leq n$, where c'_i is either c_i or a sub-plan of c_i,

   (b) or

   $$\begin{array}{l} C' = \mathrm{Case} \\ \quad\quad p_{j_1,1}, \ldots, p_{j_1,m_{j_1}} \rightarrow c'_{j_1} \\ \quad\quad \vdots \\ \quad\quad p_{j_k,1}, \ldots, p_{j_k,m_{j_k}} \rightarrow c'_{j_k} \\ \quad \mathrm{Endcase} \end{array}$$

   where $j_1, \ldots, j_k$ is a non-empty subsequence of $1, \ldots, n$ $(1 \leq j_1 < \ldots < j_k \leq n)$ and c'_{j_i} is either c_{j_i} or a sub-plan of c_{j_i},
4. C is an arbitrary conditional plan of the form $C = C_1; \ldots; C_n$ where C_i is either a simple plan or a case plan then $C' = C'_1; \ldots; C'_n$ where C'_i is a sub-plan of C_i for $1 \leq i \leq n$. □

Definition 7.2 *[Compact conditional plans]* A conditional plan C w.r.t. (G, s) (where G is a goal and s is a state) is said to be *compact* if no sub-plan of C achieves G from s. □

We now proceed towards defining minimal cost conditional plans. First, we define the unfolding of a conditional plan with respect to a complete state. (Note that it is different from unfolding a control module.) Intuitively it is the sequence of actions that will be executed if the conditional plan is executed in that situation. We then define an ordering between conditional plans which is based on comparing the cost of all the different unfolding of the two plans. We then use this ordering to define minimal cost conditional plans. The following three definitions formalize the above intuition.

Definition 7.3 *[Unfolding of Conditional Plans]* Let s be a complete state. For a conditional plan C,

- If C is empty or just an action then $Unfold(C,s) = C$.
- If C is of the form $c_1; c_2$, then $Unfold(C,s) =$ $append(Unfold(c_1,s), Unfold(c_2, \Phi(Unfold(c_1,s),s)))$.
- If c is case plan of the form given in Definition 5.1 then
  1. if one of the $p_{i,1}, \ldots, p_{i,n_i}$ are true w.r.t. s then $Unfold(C,s) = Unfold(C_i,s)$, or
  2. if none of the $p_{i,1}, \ldots, p_{i,n_i}$ are true w.r.t. s then $Unfold(C,s) = []$. □

Definition 7.4 *[Ordering between Conditional Plans]* Let S be a set of complete states, and C_1 and C_2 be two conditional plans which achieve G from a possibly incomplete state s. We say

1. $C_1 \leq_{S,s} C_2$ if for all s' in S which is a complete extension of s, $cost(Unfold(C_1,s')) \leq cost(Unfold(C_2,s'))$.
2. $C_1 <_{S,s} C_2$ if $C_1 \leq_{S,s} C_2$ and there exists a complete extension s' of s such that $cost(Unfold(C_1,s')) < cost(Unfold(C_2,s'))$. □

In the following, given a set of states S, $s \in S$, and two plans achieving a goal G from s, we write $C_1 \leq_{S,s} C_2$ (or $C_1 <_{S,s} C_2$) iff $C_1 \leq_{S^*,s} C_2$ (or $C_1 <_{S^*,s} C_2$) where S^* is the set of complete states in S.

Definition 7.5 *[Minimal Cost Conditional Plans]* Let S be a set of states. A conditional plan C achieving G from a state s is called a minimal cost conditional plan (or minimal cost plan) achieving G from s if C is a compact conditional plan w.r.t (G,s) and there exists no conditional plan C' which achieves G from s such that $C' <_{S,s} C$. □

We are now almost ready to state the sufficiency conditions. We first state the completeness condition.

Definition 7.6 *[Completeness]* An achievement control module M is said to be *complete* w.r.t. a set of states S, if for each s in S there exists at least one rule in M which is applicable in s. □

Recall that in Section 4.1, the soundness condition involved having an arbitrary prefix of the minimal cost plan in the RHS of control rules. Since we are now dealing with conditional plans and incomplete states a straight forward extension of the soundness condition in Section 4.1 does not work. (Example 13.1 in Appendix D illustrates a counter example.) Instead of an arbitrary prefix, we need to consider a special prefix – the sequence of actions of a plan until its first case plan. The next definition formally defines this special prefix of a plan.

Definition 7.7 *[Special prefix of conditional plan]* Given a conditional plan C, the special prefix of C, denoted by $pref(C)$, is defined inductively as follows:

1. If C is the empty plan $[]$ then $pref(C) = []$.
2. If C is an action a then $pref(C) = a$.
3. If $C = a_1; \ldots; a_n; P$ where P is a case plan then $pref(C) = a_1; \ldots; a_n$.
4. If $C = C_1; C_2$ and C_1 does not contain a case plan then $pref(C) = pref(C_1); pref(C_2)$.

5. If $C = C_1; C_2$ and C_1 does contain a case plan then $pref(C) = pref(C_1)$. □

Before defining soundness, we would like to define the notion of knowledge-preserving actions, which we use in the definition of soundness. Intuitively we say an action (sensing on non-sensing) a is knowledge-preserving in a state s, if by executing that action we do not lose knowledge. I.e., if a fluent f has a truth value $true$ or $false$ in s, then after executing a the truth value of f should not become unknown. Even though we do not have specially designated knowledge losing actions, a simple non-sensing action may some times result in the loss of knowledge. For example, consider a state of the robot where the robot knows f to be true and does not know the value of g. Suppose we have an action a which causes f to be false if g is true. Now if the robot executes a, it will no longer know – through its reasoning, but without sensing – what the value of f will be. This is because in the real world g could be true. But the agent does not know one way or other and hence can not be sure if f remains true after the execution of a, or if f changes its value.

Definition 7.8 *[Knowledge-preserving actions and plans]* We say that an action a in an action theory A is knowledge-preserving in a state s if for every ef-proposition of the form a **causes** f **if** $p_1, \ldots, p_n$ in A, either $p_1, \ldots, p_n$ are all true w.r.t. s or at least one of them is false w.r.t. s. Similarly for every k-proposition of the form a **determines** f **if** $p_1, \ldots, p_n$, in A, either $p_1, \ldots, p_n$ are all true w.r.t. s or at least one of them is false w.r.t. s.

We say P is a knowledge-preserving plan in a state s, and with respect to a goal G, if during the execution of P in state s, for any intermediate state s', the action executed next is knowledge-preserving in s'. □

We now define the soundness condition.

Definition 7.9 *[Soundness]*

1. A simple control rule r is said to be *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if for all $s \in S$ such that r is applicable in s there exists a minimal cost (w.r.t $\leq_{S,s}$) knowledge-preserving[18] plan C w.r.t (G, s) that achieves G from s and
   (a) if s is an incomplete state then $RHS = pref(C)$, and
   (b) if s is a complete state then the RHS of r is a nonempty prefix of $pref(C)$.
2. A termination control rule r is said to be *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if its LHS satisfies G.

An achievement control module M is *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if each rule $r \in M$ is sound w.r.t (G, S). □

The following propositions lead us to the main result of this section. The next proposition formalizes that by using only *knowledge-preserving actions* the agent never looses knowledge; i.e. if it knows the truth value of a fluent at an instant, it will continue knowing its truth value in future, although the truth value may change. The Proposition 7.2 states that the special prefix of plans with case plans always have a sensing action. These two propositions

[18] Only knowledge-preserving plans are considered and then the minimal cost criteria is applied.

are used in proving the main theorem of this section that states that control modules satisfying the sufficiency conditions are correct. The propositions are used particularly in showing that the unfolding of a sound and complete control module results in a finite conditional plan.

Proposition 7.1 Consider an action theory A and a state $s = \langle T, F \rangle$ in the language of A. Let a be an action that is knowledge-preserving in s. Then, for every state $s' = \langle T', F' \rangle$ in $\Phi(a, s)$ we have that if $f \in T \cup F$ then $f \in T' \cup F'$.

Proof: Follows directly from the definition of knowledge-preserving actions, and the fact that the action a is knowledge-preserving in s. □

Proposition 7.2 Let S be a set of states and C be a compact conditional plan that achieves G from s. If C contains a case plan and $pref(P)$ is knowledge-preserving in s then $pref(P)$ contains a sensing action, and s is an incomplete state.

Proof: In Appendix D. □

Theorem 7.1 Let A be an action theory. Consider a pair (M, S), where S is a set of (possibly incomplete) states, M is a control module (possibly with sensing actions) sequential w.r.t. S, and S is closed w.r.t. M. Given a goal G, if M is sound w.r.t. G and S and complete w.r.t. S then M achieves G from S w.r.t. A.

Proof: In Appendix D. □

Using the above theorem we can now easily show that the control module *Open_Door* achieves the goal *door_open* with respect to the causal rules of *Open_Door*, from all states where the truth value of *door_open* is either true or false.

We now give the sketch of an algorithm that uses Theorem 7.1 to construct a control module that achieves a goal G from a set of states S.

Algorithm 7.1

Input: Goal G, a set of states S, an action theory for the robot and for the exogenous actions.

Output: A Control Module M that achieves G from S.

Step 1 $M = \{\mathbf{if}\ \ G\ \mathbf{then}\ \ HALT\}$,
$S_{in} = \{s' : s'$ is reachable from some state s in S by applying sequences of exogenous actions$\}$, and
$S_{out} = \emptyset$.

Step 2 While $S_{in} \neq S_{out}$

2.1 Pick a state s from $S_{in} \setminus S_{out}$

2.2 *Find a minimal cost knowledge-preserving plan*[19] P from s that achieves G. Let a be the first action of P.

2.3 If P is a non-null plan and $s = \langle\{a_1, \ldots, a_n\}, \{b_1, \ldots, b_m\}\rangle$ is incomplete then $M = M \cup \{r\}$ where r is the rule:
$\mathbf{if}\ \ a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m, u(c_1), \ldots, u(c_k)\ \mathbf{then}\ \ pref(P)$;
(where $c_1, \ldots, c_k$ are the remaining fluents in our world)
else if P is a non-null plan and s is a complete state then $M = M \cup \{\mathbf{if}\ \ s\ \mathbf{then}\ \ A\}$;

2.4 $S_{in} = S_{in} \cup [a]s \cup \{s' : s'$ is reachable from $[a]s$ by applying sequences of exogenous actions$\}$ and $S_{out} = S_{out} \cup \{s\}$.

Step 3 Merge control rules in M which have the same RHS and whose LHS can be combined.

□

Proposition 7.3 Given a goal G, a set of states S and an action theory A if G is achievable from all states in $Closure(S, A, A)$ through knowledge preserving plans, then the above algorithm generates a control module M such that M achieves G from $Closure(S, M, A)$.

Proof:(sketch)
We are assuming that our theory has exogenous actions that add knowledge. Hence after initializing the closure in step 1 we compute it iteratively in Step 2.4. In step 2.3 we add sound control rules for the state picked from S_{in}. Since the algorithm terminates when $S_{in} = S_{out}$, the constructed control module is also complete. Hence by Theorem 7.1 the algorithm computes a control module that achieves the goal G from the set of states $Closure(S, M, A)$. □

The complexity of the above algorithm also depends on the size of the closure and the complexity of finding conditional plans. One difference from the previous algorithm is the fact that the number of states in the worst case will be 3^n – where n is the number of fluents – if we use the approximate theory of sensing in [BS97]. It will be much larger 2^{2^n} if the more general theories of sensing in [Moo85, SL93, LTM97] is used. It seems to us that the conditional planner in [GW96] uses the theory of sensing in [BS97].

8 Our Agent theory – Reactivity and Deliberativity

In the last two sections we formalized the correctness of control modules, gave sufficiency conditions that guarantees correctness, and gave an algorithm that uses the sufficiency conditions to construct correct control modules. *We reiterate that control modules are verified for their correctness and/or automatically constructed off-line; not during the execution.* In essence, we advocate off-line deliberations to create or verify control modules that are reactive on-line.

But, we distinguish our control modules from universal plans by requiring the control module to be correct with respect to a set of initial states and additional states – not all possible states, that can be reached from these initial states by executing the given control module, or through occurrences of exogenous actions as dictated by its theory. The intuition behind this approach is that, to be within manageable space requirements the control module should only be applicable with respect to a set of plausible states.

We advocate that if the agent happen to get into a state that was not considered plausible and thus was not taken into account during the construction of the control module, then it should deliberate on-line to create a plan to reach one of the accounted for states – the states in the closure. Note that we do not require that a plan to reach a goal state should be found. We hope that since the set of accounted for states is much larger than the set of goal states, it will be easier and faster to find a plan that takes us to one of those states. From there, the control module will take the agent to one of the goal states.

Overall, our agent control can be summed up as off-line deliberation for on-line reactivity from a set of plausible states, and on-line deliberation as a backup to take the agent from a rare unaccounted for state to one of the accounted for states.

9 From theory to practice: Our robot in the AAAI 96 contest

We participated[20] in the AAAI 96 robot navigation contest [KNH97]. *Our team scored 285 points in the contest out of a total of 295 points and was placed third.* Our mobile robot entry was also in the first place in the finals of the home vacuuming contest in AAAI 97. In this section we briefly discuss the top level control of our mobile robot program that is directly related to the theory discussed so far. A more detailed account of our robot entries is presented in [BFH$^+$98].

In the AAAI 96 robot navigation contest [KNH97] robots were given a topological map of an office like environment and were required to achieve a particular navigational task. In particular the robot was required[21] to start from the directors office, find if conference room 1 was available (i.e., empty); if not, then find if conference room 2 was available; if either was empty, then inform professor1, professor2 and the director about a meeting in that room; otherwise inform the professors and the director that the meeting would be at the director's office, and finally return to the director's office. Robots were required to do all this without hitting any obstacle, and without changing the availability status of the conference rooms.

Our top-level module was a control module of the kind described in Section 5.1. Its reactive structure was geared towards gracefully recovering

[20] Other active members of our team were David Morales, Monica Nogueira, and Luis Floriano. We were also assisted by Alfredo Gabaldon, Richard Watson, Dara Morganstein and Glen Hutton.

[21] To focus on the main point we have simplified the real requirement a little bit.

from breakdowns which can be modeled as exogenous actions. Following was our top-level control module:

if $\neg visit\_conf\_1$ **then** $go\_to\_conf(1)$
if $at\_conf(1), u(avail(1))$ **then** $sense\_avail(1)$
if $at\_conf(1), \neg avail(1)$ **then** $go\_to\_conf(2)$
if $at\_conf(1), avail(1), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_conf(2), u(avail(2))$ **then** $sense\_avail(2)$
if $at\_conf(2), avail(2), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_conf(2), \neg avail(2), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_prof(1), \neg visit\_prof(2)$ **then** $go\_to\_prof(2)$
if $at\_prof(2), \neg back\_to\_director$ **then** $go\_to\_director$
if $back\_to\_director$ **then** $HALT$

We constructed the above control module manually. *But we were able to use extensions of the theory described in the previous section to verify the correctness of our control module.* The reason we needed to extend the theory of the previous sections was because in our theory, goals are a set of fluents. To express the goal of this control module we needed additional expressibility using temporal and knowledge operators.

We now give a declarative representation of the required goal in the AAAI 96 contest in an extension of the language FMITL (First-order metric interval temporal logic) [BK96]. Our extension allows specification of knowledge. The meaning of various operators and atoms in the following specifications are: Ka means a is known to be true; avail(1) means conference room 1 is available, informedprof1(1) means professor 1 has been informed that the meeting will be in conference room 1, $\Box f$ means always f is true, $\diamond f$ means eventually f is true, and $at(dir)$ means the robot is at the directors office.

$$
\begin{aligned}
&(Kavail(1) \vee K\neg avail(1))\wedge \\
&(K\neg avail(1) \Rightarrow (Kavail(2) \vee K\neg avail(2)))\wedge \\
&(Kavail(1) \Rightarrow (informedprof1(1)\wedge \\
&informedprof2(1) \wedge informeddirector(1)))\wedge \\
&((Kavail(2) \wedge K\neg avail(1)) \Rightarrow (informedprof1(2)\wedge \\
&informedprof2(2) \wedge informeddirector(2)))\wedge \\
&((K\neg avail(1) \wedge K\neg avail(2)) \Rightarrow (informedprof1(dir)\wedge \\
&informedprof2(dir) \wedge informeddirector(dir)))\wedge \\
&\forall X avail(X) \Rightarrow \Box avail(X)\wedge \\
&\Box clear\_from\_obstacle\wedge \diamond\Box at(dir)
\end{aligned}
$$

Extending our definition of correctness (in Definition 6.4) to such goals is straight forward. We just need to check that the unfoldings satisfy the goal. Due to temporal operators in the goal the trajectory of the states becomes as important as the final state. Because of this our sufficiency conditions and the automatic construction algorithm based on them are not directly applicable. *One of our future goals is to find algorithms to automatically generate control modules for complex goals with knowledge and temporal operators.*

9.1 Modules to take from one room to another room – The navigation module

The top level control module used two sensing actions: $sense\_avail(1)$ and $sense\_avail(2)$, and five non-sensing actions: $go\_to\_conf(1)$, $go\_to\_conf(2)$, $go\_to\_prof(1)$, $go\_to\_prof(2)$, and $go\_to\_director$. The two sensing actions were implemented using sonars that checked if there was any substantial movement in the room, and did not have a reactive structure. The non-sensing actions were abstracted as going from one landmark (or node) in the topological map to another. This was implemented using a control module. We now describe this control module, which we refer to as the navigation control module.

The navigation control module was basically a table with three columns: current node, goal node, next adjacent node. An entry (5,3,4) in that table meant that if the robot is currently at node 5 and its ultimate goal is to reach node 3 then it should next go to the adjacent node 4. The actions in this module were actions that took the robot from any node to its specified adjacent node. *This control module was automatically generated and we used the approximate distance between adjacent nodes as the cost of the action that takes the robot from one of those nodes to the other.*

The actions that took the robot from one node to one of its specified adjacent node was also implemented by a control module. We now describe this control module.

9.2 Control module for going to an adjacent node

The actions used in this control module were of two types: turning actions, that changes the heading of the robot and navigational actions, that takes the robot from one kind of node to an adjacent node of another kind. The turning actions are listed in the third column of Table 2 and the navigation actions are listed in the third column of Table 1.

Type of From Node	Type of To Node	Action
corridor	junction	*detect_break*
junction	corridor	*detect_corridor*
X	Y, Y=X	*go_forward*
junction	door	*go_to_door*
door	junction	*go_to_door*

Table 1: Some action rules for generation of navigation actions

Following gives a flavor of how the effect of navigation and turning actions could be formally specified:

$detect\_corridor$ **causes** $at(X), \neg at(Y)$**if** $at(Y), type(Y, junction),$
$type(X, corridor), heading(dir), adj(X, Y, dir)$

$turn\_180$ **causes** $heading(west), \neg heading(east)$**if** $heading(east)$

In the above specification, $type(X, corridor)$ means that node X is of the type corridor. Similarly, $adj(X, Y, dir)$, means that the nodes X and Y are adjacent and to go from node X to node Y the robot's heading must be dir. We now give a simple map with five nodes and give a table that has the adjacency and heading information for the given map. Intuitively, the first row of Table 3 can be read as node 5 is of type corridor, node 4 is of type junction, nodes 5 and 4 are adjacent, approximate distance between them is 150 units, and to go from node 5 to node 4 the robot's heading must be west.

Now let us get back to the control module that takes the robot from one node to a specified adjacent node. The condition part of each control rule in

Current Heading	Need to be	Action
east	west	*turn_180*
east	south	*turn_right_90*
X	Y, Y=X	
east	north	*turn_left_90*
west	east	*turn_180*
west	south	*turn_left_90*
west	north	*turn_right_90*
north	west	*turn_left_90*
north	east	*turn_right_90*
north	south	*turn_180*
south	east	*turn_left_90*
south	west	*turn_right_90*
south	north	*turn_180*

Table 2: Action rules for generation of turning actions

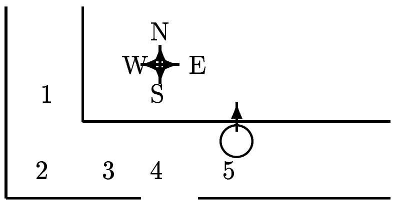

Figure 2: A sample map.

the control module checks if the robots heading is appropriate (as specified in Table 3), and if not, the robot executes the turning action specified by the Table 2. If the heading is appropriate the robot executes the navigation action specified by Table 1. In other words, Tables (1 and 2) are compiled from the description of actions and describe which action to take given particular information about an initial state and a goal. *Tables (1 and 2) were generated manually and their correctness was verified using the theory of the paper.*

N1	Type N1	N2	Type N2	Heading	Dst.
5	corridor	4	junction	west	150
4	junction	3	corridor	west	100
3	corridor	2	wall_break	west	150
2	wall_break	1	corridor	north	150

Table 3: An example showing adjacency records for the domain depicted by Figure 2.

9.3 Basic actions and Skills

We refer the actions in the previous control module (the actions in the third column of the tables 1 and 2) as the basic actions. They were implemented by *merging* several skills which were implemented using *control modules based on multi-valued logic.* At the lowest level, the uncertainty associated with the physical world makes it harder to implement low level tasks using control modules of the kind (where, fluents are 2-valued) described in this paper. We found the control modules of the kind described in [SKR95] that uses multi-valued logic to be more appropriate for that purpose. We also used the concept of 'merging' control described in [SKR95] for blending skills into basic actions. For example, the basic action of going forward means translating forward with a certain speed and at the same time avoiding obstacles, and avoiding a crash. I.e., the basic action go_forward involves the *blending* (or *normalization*) [SKR95] of many of the skills (not necessarily all) with respect to assumed associated skill priorities: *Avoid Crash, Control Steering, Control Speed, Facing Ahead, Avoid Obstacles, Search for Destination.* The tables 4 and 5 list the skills, and basic actions of our robots, respectively, and their intuitive function.

Skill	Function
Translate_Forward	Always traverse forward.
Point_To_Opening	Find openings for the robot to traverse through.
Follow_Corridor	Follow a virtual corridor and look for a hallway of a given width.
Avoid_Close_Sonar	Avoid close sonar sensor readings.
Avoid_Close_InfraRed	Avoid close infra red sensor readings.
Stop_When_Touched	Stop when bumped.
Avoid_Crashing	Stop if in danger of crashing into obstacles.
Point_Forward	Keep robot from turning back.

Table 4: Skills implemented in Diablo.

Action Name	Effect
go_forward	Go forward for a given distance.
detect_corridor	Go forward until a corridor of a given width is detected.
go_to_door	Go forward some distance to enter a room.
detect_break	Follow a corridor of a given width until a change in the corridor width is detected.
turn_left_90	Turn left 90 degrees.
turn_right_90	Turn right 90 degrees.
turn_180	Turn 180 degrees.
speak	Transmit information.
detect_occupied	Sensing action to detect if a room is occupied.
align	Correct hardware uncertainty.

Table 5: Basic actions

Although, Saffioti et al [SKR95] have a notion of correctness for their control modules, they do not consider exogenous actions, and sensing actions. *One of our future goal is to extend our approach to the control modules in [SKR95] in defining correctness in presence of exogenous and sensing actions, as well as developing methods to automatically construct such control modules.*

10 Related Work

In this section we relate our approach and results in this paper to other related work. In particular we compare our work with earlier research on universal plans, situation control rules, robot execution languages, and agent theories and architectures.

10.1 Universal plans and situation control rules

Control modules as formulated in this paper have similarities with universal plans [Sch87] and triangle tables [Nil85] in the sense that *for a set of situations* they specify what the robot should do in each situation. *The key difference* is that Universal plans [Sch87, Sch89b, Sch89a, Sch92b] prescribe what actions need to be executed in *each possible situation. In contrast* our control modules *only consider the closure of a given set of initial states.* If we consider the initial set of states to be the set of most likely states that the robot might be initially in, and the action theory about the exogenous actions to encode the most likely exogenous actions that may occur, then the closure is the set of states the robot is most likely to be in. This set of states could be fairly small compared to the set of all possible states. *Another important aspect* is that when defining the closure we are very careful in considering states reached from the initial state through the actions of the robot. Since the robot has control of its own actions, we don't need to consider all possible states that can be reached by *some arbitrary* sequence of the robot's actions. *We only need to consider those sequence of actions that are dictated by the control module.* For these reasons, the concept of closure and its fixpoint construction – described in section 4.2 – are important.

In contrast to the approach in universal plans, which is sometimes criticized for its intractability [Gin89, JB96][22], our control modules are *carefully selected* subset (the closure) of the universal plan, which gives reactivity to the robot. Our view is that, to act in rare situations not in this carefully selected set, the robot can make a plan in real time to one of the set in the closure (not necessarily only to situations satisfying the goal); as the robot knows how to go from there to a situation satisfying the goal conditions.

The sufficiency conditions about our control modules *guarantee* the correctness of the control module – thus strengthening the results in [Sch87, Dru89]. In particular:

[22] Ginsberg in [Gin89] argues that almost all (interesting) universal plans take an infeasibly large amount of space. Jonsson and Backstorm [JB96] formally show that universal plans which run in polynomial time and are of polynomial size can not satisfy the condition that – if the problem has a solution, then the universal plan will find a solution in a finite number of steps. On the other hand Schoppers in [Sch94, Sch95] shows why the expected state-space explosion in Universal plans does not happen in many realistic domains where the fluents are often not independent of each other. Another formal treatment of universal plans is done in [Sel94].

- Schoppers in Section 6.1 of [Sch87] says, "Universal plans not only anticipate every possible situations in a domain but actually prescribe an action for every initial state; more over the prescribed action is *usually optimal*.
  In our formulation the prescribed action is optimal.
- Drummond in [Dru89] says, "sound SCRs guarantee that local execution choices always lead to *possible goal achievement*".
  In our formulation, local execution choices always lead to goal achievement.

  Drummond and his colleagues' later work [DB90, DBS94] has a lot in common with our approach here. In [DB90], they present an algorithm for incremental control rule synthesis. In [DBS94], they present an algorithm for building robust schedules that takes a nominal schedule and builds contingent schedules for anticipated errors. They validate their algorithm experimentally in a real telescope scheduling domain. But in these works *they do not have a formal correctness result about their algorithms.* In this paper, our notion of closure, our notion of an action theory for exogenous actions, and our notion of correctness with respect to the closure, precisely formulates their idea of 'contingent schedules for anticipated errors'. Moreover our algorithms guarantedly construct correct control modules, and we also experimentally validate our approach in the domain of a mobile robot in an office environment

We must note that the use of shortest plans by Jonsson and Backstorm in constructing universal plans in the proof of Theorem 10 in [JB96] is similar to the minimal cost condition in our definition of soundness of control rules. Our initial research and [JB96] were done independently around the same time. The minimal cost condition is of course more general than the shortest path condition.

Kaelbling and Rosenschein [KR91, RK95] were one of the early researchers working on 'situated agents' who were also interested in 'representation', and any formal connection between them. In [KR91], they say that in a control rule 'if $p_1, \ldots, p_n$ then a', the action a, must be the action that leads to the goal. *In this paper we formalized what it means by 'an action leading to a goal'.* One of the main difference between the approach in [KR91] and here in terms of automatic construction of control modules is that in [KR91] control rules are obtained not from action theories (as in this paper) but from a hierarchical description of how to achieve the goal. In other words while this paper is based on the STRIPS approach to planning, the approach in [KR91] is based on the HTN approach to planning.

In [Nil94], Nilsson presents a formalism for computing and organizing actions for autonomous agents in dynamic environments. His condition actions rules are similar to our control rules, and he also allows hierarchy of control programs. *Nilsson says that his presentation of control programs is informal as he believes that formalization is best done after a certain amount of experience has been obtained.* After working with mobile robots, this paper is our attempt at formalization of control programs. Also although we did not stress on it earlier, as in [Nil94] we allow variables in control rules; for example in the control module *Goto_elevator_2*.

Recently Saffioti et al [SKR95] formalize the notion of correctness of control modules which use multi-valued logic to take into account uncertainty of the environment and sensor noise. We do not take into account these concerns and as a result our approach is not as adequate for low level control of robots in certain environments. On the other hand, they do not consider exogenous actions, and more important, they do not give any sufficiency conditions for correctness, nor any procedure to automatically construct control modules.

Finally in this paper we do not use any specific action theory, and avoid the debate about which action theory is better; rather we use an abstract 'entailment relation'. Because of this, our formulation need not change with more sophisticated action theories.

10.2 Reactive planning, Program synthesis and Discrete event dynamic systems

Recently there has been some proposals [GK91, KBSD97, DKKN95] about automatic construction of control rules inspired by research in program verification and synthesis and operations research. In [KBSD97], an algorithm to generate control rules for goals given in a temporal logic is given. Two major contributions of this work are that it considers deadlines, and it allows general temporal goals that can specify cyclic behaviors. In [DKKN95], transitions between states are represented as Markov processes, and goals are specified using reward functions, and policy iteration algorithms from Operations Research are used to construct control rules. Our approach differs from the approach in [KBSD97, DKKN95], the approaches mentioned in the program verification and synthesis literature (for example, [Eme90, EC82, PR89]), and the approaches in the discrete event dynamic systems literature (for example, [OWA91, RW87a, RW87b]), in that we separate agent actions from exogenous actions and do not combine them and our notion of maintainance is weaker than analogous notions there. Also we insist on developing (sufficiency) conditions for the correctness of *individual* control rules. The formulations in [KBSD97, DKKN95] and the ones mentioned in [Eme90] only deal with the correctness of a control module as a whole. It is important to consider correctness of individual control rules, because often we *learn* (or we are told) a particular control rule in isolation, and we need to satisfy ourselves that it is correct by itself, regardless of the rest of the control rules. Also our methodology allows us to upgrade a control module by simply adding additional correct rules for new states that need to be taken care of. In case of [KBSD97, DKKN95], simply adding new rules may not be always enough to upgrade a module and extra care is needed to avoid getting into the kind of cycles present in the module Goto_elevator_3 from Section 4. Finally we consider sensing actions which are not considered in [KBSD97, DKKN95, OWA91], and we use AI methodologies such as 'planning' which is not considered in the program reasoning approaches described in [Eme90, EC82, PR89].

10.3 Agent theories and architectures

Our approach in this paper has been to use action theories to formalize correctness of agents whose control is represented as an hierarchy of control modules, and to develop methods to construct such control modules. There has been a lot of work on agents, some of which are directly based on action theories. In the following subsection we briefly discuss this research and compare them to our work. In a later subsection we give a brief description of the agent architectures that use action theories.

10.3.1 Agent theories and architectures not directly based on action theories

A starting point on recent research on agents is the collections, Intelligent Agents I, II and III [Woo94, WMT95b, MWJ96], the proceedings of Autonomous Agents 97 [Joh97], the collection, [Mae91a], proceedings of MAAMAW workshops [vdVP96], and the monographs [Mul96, Sin94]. Additional pointer includes, Woolridge et al. [WMT95a], which contains a fairly exhaustive bibliography of papers on agent theories, architectures and languages.

In the background section of his monograph [Mul96], Muller roughly classifies agent architectures to four categories: deliberative agents, reactive agents, interacting agents, and hybrid agents; and gives a nice overview of work in each. We will now compare our approach in this paper with each of the above four classes, as classified in [Mul96].

The closest to the approach in this paper are the *reactive agents* [Bro86, AC87, KR91, Mae91b], and our whole paper is about formulating correctness of such reactive agents. Although, we have used a particular definition, where a control module consists of situation-action control rules of a particular kind [KR91], our formulation can be easily adapted to the other related variants, such as the subsumption architecture [Bro86]. Moreover, we present algorithms to automatically construct control modules, and allow special sensing actions in our control modules, not considered in the earlier approaches.

The conventional approach in *hybrid architectures* [BKMS95, Fir92, Gat92] is to use reactivity at the lower levels and deliberativity at the higher levels. *In this paper we argue that on-line reactivity is due to tables compiled during off-line deliberations.* Nevertheless, our approach is hybrid in a different sense – reactivity for anticipated highly plausible situations (situations in the closure set), while occasional deliberativity for rare situations that have not been taken into account during compilation.

Our formulation of correctness of reactive modules using action theories, relates reactivity with deliberativity. But this does not elucidate the connection between our approach and the *deliberative agents* [RG95, KGR96, ST92, Tho95, Bra87] mentioned in [Mul96]. In [RG95, KGR96, Bra87] agents are supposed to be rational and have, as said in [RG95], "mental attitudes of Belief, Desire, and Intention (BDI), representing, respectively, the information, motivational, and deliberative states of the agent." Such agents have been formalized using temporal logics [EJ89] together with modal operators for beliefs, desires and intentions [RG91, CH90]. We see two connections between these approaches and our approach;

- The formulation of agent behavior using the temporal logic CTL* and its extensions [Sin94, RG91] represents worlds as time trees. These time trees are similar to the transition function used in action theories; the main focus in the later being developing *elaboration tolerant* languages to specify actions and their effects, and the nature of the world, and characterizing these languages.

  Moreover, although, in the early stages goals were collection of fluents, complex goals in an action theory setting, as described in [BK96] use temporal operators and are very similar to propositions in CTL*.

- When looking for the BDI aspect of an agent based on the approach of this paper, it is clear that, *beliefs* correspond to the action theory and the current values of the fluent and *desires* correspond to the goal. The counterpart of *intention* in our formulation is not that obvious. *We believe that intention in a particular situation corresponds to the plan obtained by unfolding the control module with respect to that state.*

With this correspondence as the starting point, it seems that that we can have finer notions of intentions. For example, *immediate intention* in a situation corresponds to the actions in the RHS of control rule applicable in that situation. But, since our actions can themselves be defined using control modules, we can have finer definitions of immediate intention depending on to what level we decompose our actions. Similarly, we can define *n-action intention* at a situation, as the next n actions that the agent will be doing (this again can be obtained by unfolding the control module) if no exogenous actions happen. This can be further generalized to *n-action intention w.r.t. the exogenous action theory* A at a situation, as a n-level decision tree of robot actions and exogenous actions, describing what the agent would be doing under different occurrences of exogenous actions based on the theory A. Moreover, in presence of sensing actions, an *n-action intention* at a situation, could be a conditional plan, as in presence of sensing and incompleteness, the unfolding may produce a conditional plan. We discuss this correspondence in further detail in [BG97b].

There has been a lot of work on *interacting agents* [Fis93, Jen92, SBK$^+$93, BS92]. Although, we allow exogenous actions, we have not touched upon multiple cooperating agents in this paper. We hope to expand our ideas in this paper to such a case. We believe that in our future work along these lines we will be using the ideas of multi-agent planning [Lan89] and multi-agent action theories [Moo85].

10.3.2 Agent theories and architectures based on action theories

In the introduction of this paper we referred to several papers on action theories. In this section we consider three aspects of action theories (complex actions, sensing actions, and narratives) and their impact on agents.

The Cognitive Robotics group at the University of Toronto have developed several robot execution languages, GOLOG (alGOl in LOGic) [LLL$^+$94, LRL$^+$97], CONGOLOG [LLL$^+$95, DGLL97], and $\mathcal{R}$ [Lev96] to specify the execution program of a robot. GOLOG allows specification of complex actions as macros and has constructs such as: conditional statements, non-deterministic choice of actions and action arguments, non-deterministic iterations, recursive procedures, etc. A GOLOG program when executed uses an extended version of situation calculus to simulate the changes in the world so as to decide on the executability of an action before actually executing it, and also to decide which branch to take when faced with a conditional statement. The group at Toronto have developed several interpreters of GOLOG, mainly written in PROLOG. The off-line interpreter verifies the executability conditions, evaluates the conditions in the conditional statements, and makes choices at the non-deterministic choice points, before actually executing the program. The correctness of a GOLOG program with respect to a goal can be verified by adding a special action at the end of the program and setting its executability condition as the goal. The similarity between [LRL$^+$97] and our approach is that both formalize a notion of correctness of complex robot execution programs. Note that in our approach a control module may be considered as a complex robot execution program. Besides this similarity, there are several differences[23]. GOLOG allows non-deterministic actions, and recursive procedures while we do not. On the other hand GOLOG does not consider exogenous actions, does not allow sensing actions and hence is not suitable for an agent in an environment where changes beyond the control of the robot may occur. This means that not only it is not reactive, but also that it can not do deliberative reasoning based on observations about the dynamic world. All the above criticisms are also applicable to $\mathcal{R}$ which does allow sensing actions. CONGOLOG [LLL+95] (the most recent version appears in [DGLL97]) is an extension of GOLOG that allows concurrent execution with priorities and interrupts that are very much like control rules. But it does not allow sensing actions and its simulated account of exogenous action may not match the real world. Finally while GOLOG, CONGOLOG and $\mathcal{R}$ are more structured, their focus is not about the automatic construction of robot programs, which is one of our main concerns. The simple structure of our language makes it easier for us to automatically construct programs in our language.

Recently in [DRS98], a notion of 'execution monitoring' has been added to GOLOG. With execution monitoring their robot can now observe the world and make plans to recover from states reached due to exogenous actions from where the original GOLOG program is no longer executable. Although such robots can now deal with exogenous actions, they are still not reactive (as planning to recover may take substantial time). Moreover the approach in [DRS98] can not take advantage of 'opportunities', that may be sometimes provided by exogenous actions. But both these shortcomings can be easily avoided by incorporating some kind of execution monitoring to CONGOLOG. Still these extensions do not consider special sensing or sensing actions. (Recently, in the 1998 AAAI Fall symposium two different extensions of GOLOG and CONGOLOG have been presented that allow sensing actions.)

Many theories of actions (including GOLOG and CONGOLOG) do not allow specification of observations and execution of exogenous actions. Hence they are not able to adequately capture dynamic worlds. Recently some action theories have been proposed [PR93, MS94, BGP97, DRS98] that allow specification of observations and action executions. These theories are adequate to represent dynamic worlds. In [BGP97], an architecture for autonomous agents in a dynamic world has been given. This architecture can be used *to make plans from the current situation* – thus taking into account changes that happened since the last plan was constructed. A similar architecture is also suggested in [Kow95], although without a detailed theory of action. But the architectures in [Kow95, BGP97] are not appropriate for a reactive agent. This is because they plan and reason while the agent is acting in the world, and the agent in the dynamic world does not normally have enough time to plan and reason while acting. But under rare circumstances when the agent's reactive mechanism fails, these approaches may be used as a backup. The approach in [GEW96], which also does planning and execution in real time will not be reactive and not normally appropriate for an agent that needs to react quickly, particularly in the presence of exogenous actions. Wagner in [Wag96] and Li and Pereira [LP96] propose to use both action theories and reaction rules to develop agents that are both re-

[23] These differences are dynamic in the sense that the research in Toronto is ongoing and they are already working on many of the aspects we discuss.

active and deliberative. But they do not consider correctness aspect of the reactive part, do not allow sensing actions, and do not consider automatic generation of the reactive part.

Finally most work on universal plans and control programs have not considered sensing actions together with a formal theory of knowledge. One of the exceptions is [Sch95]. In page 183 of that paper Schoppers discusses how his work bridges the gap between research in reasoning about knowledge and action, research in plan modification that verified progress during plan execution, and research in situated agency with execution-time sensing. The formal results about correctness and automatic construction of control rules with sensing actions in this paper augment the results of [Sch95]. In our formulation where we use sensing actions, we greatly benefited from [Moo85, SL93, EHW$^+$92, Lev96, GEW96, GW96]. The main focus in these papers is how to proceed towards achieving a goal in the presence of incomplete knowledge about the world. The necessity of conditional plans and knowledge producing actions for planning in presence of incomplete information was discussed in [EHW$^+$92, KOG92, PS92, Sch92a]. An initial logical account of knowledge producing actions was given in [Moo77, Moo79, Moo85] and later expanded in [SL93]. Recently Levesque [Lev96] used the theory in [SL93] to formalize correctness of conditional plans in the presence of incomplete information. The formalization of knowledge producing actions [BS97] that we use in this paper is weaker but simpler than the formalization in [SL93].

11 Conclusion and Future Directions

In this paper we formulate the correctness of reactive control modules (with or without sensing actions) with respect to a set of initial states, an action theory for the agents actions, an action theory for the exogenous actions in the environment, and a goal. One important aspect of our formulation, which is different from other related formulations in the literature [KBSD97, DKKN95], is that we exclude exogenous actions while defining the correctness with respect to a single state but take them into account in determining the the set of states the agent may get into.

We then give sufficiency conditions for the correctness of both individual control rules and control modules as a whole and use them to develop an algorithm to automatically construct correct control modules when given a goal, a set of initial states and action theories of the agent and the environment. Some of the directions we would like to extend our work are:

- We would like to consider goals that not only define the final state, but also constrain the trajectory used to reach that final state. We would also like to consider the specification of knowledge in our goals. To represent these kind of extended goals we will need temporal and knowledge operators. (Such operators are used by Schoppers in [Sch95].) Although our formulation of correctness can be easily generalized to such goals, we do not currently know what kind of sufficiency conditions we will need and how to extend our algorithms for such goals.

- We would like to extend our approach to go beyond a single agent (in a dynamic world) to a collection of multiple co-operative agents, and formalize the correctness of a set of control modules corresponding to a set of co-operative agents. Although CONGOLOG [DGLL97] is such a formalism we would like to further allow exogenous actions and also look for automatic control module generation algorithms.

- We would like to further investigate the relation between our approach and the probabilistic approach in [DKKN95] and the multi-valued logic based approach in [SKR95]. In the former we would like to study the connection between lack of knowledge expressed through POMDPs (partially observable Markov decision processes) and through logics of knowledge. As regards to the later, Saffioti et al formulate the correctness of fuzzy control modules in the absence of exogenous actions. We would like to extend their work to include the possibility of exogenous actions, and also would like to develop algorithms that will generate control modules based on multi-valued logic.

- We would like to consider horizontal and vertical combination of control modules. For example, we need horizontal combination of control modules to achieve the conjunction (or disjunction) of their goals. For quicker reactive behavior we may need vertical merging and elimination of levels of modules, where an action in a higher level module is defined by another control module. We would also like to consider our reactive module as an robot execution language and compare its expressibility with languages such as GOLOG and CONGOLOG.

- Finally the approach in this paper, where a control module is expected to be correct only w.r.t. a set of states (not all possible states), leads to an agent architecture where the agent uses the control module as a cache (similar to the idea in [Sch89a]) where reactions to a collection of important, or most plausible states – referred to as *accounted-for states* – are computed off-line and stored and when the agent gets into one of the rare unaccounted-for states, it makes an on-line plan to get to one of the accounted-for states (not just to the goal), from where it knows what to do. We plan to experimentally investigate this approach in further detail; in particular in finding how to decide which states should be accounted for, and in looking for planning algorithms that take advantage of the broader goal of reaching one of the accounted-for states.

We will conclude our paper with two quotes by Dijkstra and Hoare, respectively, on programming methodologies and relate our results in this paper to their quotes.

In [Dij72] Dijkstra says:

> The only effective way to raise the confidence level of a program significantly is to give proof of its correctness. But one should not first make the program and then prove its correctness, because then the requirement of providing the proof will only increase the poor programmer's burden. On the contrary: the programmer should let correctness proof and program grow hand in hand. ... If one first asks oneself what the structure of a convincing proof would be, and having found this, then constructs a program satisfying this proof's requirements, then these correctness concerns turnout to be a very effective heuristic guidance.

Our main goal in this paper has been to develop methodologies that will aid in the construction of *correct* control modules. In this we share Dijkstra's vision, and in fact it is more important in case of autonomous agents – which by definition are supposed to be autonomous, to be proven correct (or to be constructed correct), before they venture out on their own. Our contribution in this regard is the formalization of the notion of correctness, and the sufficiency conditions for correctness. We hope that the sufficiency conditions will at least initially serve as a guide in constructing control modules manually, and ultimately will be used to construct control modules automatically.

The formalization of correctness of control modules with respect to action theories, and the automatic construction of control modules from action theories, may sometime appear to be not very useful; particularly when the action theory itself is hard to specify. In fact in many cases, it may be easier to construct the control module than to specify the action theory. The control module $Go\_clockwise\_1room$ is perhaps such a module. But, there is a point in having alternate representations. Hoare in [Hoa87] makes this point very eloquently in the following words:

> For specifications of the highest quality and importance I would recommend complete formalization of requirements in two entirely different styles, together with a proof that they are consistent or even equivalent to each other. ... A language for which two consistent and complementary definitions are provided may be confidently taken as a secure basis for software engineering.

Finally, we would like to say that the approach in this paper can be generalized to explain the relation between 'scruffy methods' and 'formal methods' in AI and to show the importance of both. In general, in so called, 'scruffy methods' (situated automata, production systems, case based reasoning, frames, some expert systems), programs are developed that reason efficiently but the formal foundations are not clear. On the other hand in 'formal reasoning' (non-monotonic reasoning systems) the formal foundations are clear, but the reasoning is usually inefficient. *To us, the relation between the two is that the programs that do 'scruffy reasoning' are compiled (and hence are more efficient) from knowledge represented in a formal system, based on what is expected from the programs and sufficient progress in both is needed to understand the compilation process.* In this paper the 'programs that do scruffy reasoning' corresponds to the 'control modules', the 'knowledge represented in a formal system' corresponds to the 'action theory' and the 'expectation' corresponds to the expectation that the robot can act quickly after sensing. We believe this generalization will clear misunderstandings (see for example [HFA94]) between many of the researchers following either of the approaches, and will increase respect towards each other.

Acknowledgment

We are grateful to Erik Sandewall and Marcus Bjareland for useful discussions related to the subject of this paper. We are also grateful to the anonymous reviewer who pointed us to the literature in discrete event dynamic systems and the similarity between our notion of maintainance and the notion of stability there. This research was partially supported by the grants NSF CAREER IRI-9501577, NASA NCC5-97 and NSF DUE-9750858.

References

[AC87] P.E. Agre and D. Chapman. Pengi: an implementation of a theory of activity. In *Proc. of AAAI 87*, pages 268–272. Morgan Kaufmann, 1987.

[AIS88] J. Ambros-Ingerson and Steel S. Integrating planning, execution and monitoring. In *Proc. of AAAI 88*, pages 83–88, 1988.

[ALP94] J Alferes, R. Li, and L Pereira. Concurrent actions and changes in the situation calculus. In H. Geffner, editor, *Proc of IBERAMIA 94*, pages 93–104. McGraw Hill, 1994.

[Ark91] R. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In P. Maes, editor, *Designing Autonomous Agents*, pages 105–122. MIT Press, 1991.

[Bar95] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *Proc. of IJCAI 95*, pages 2017–2023, 1995.

[Bar96] C. Baral, editor. *Proceedings of AAAI 96 Workshop on Reasoning about actions, planning and robot control: Bridging the gap.* AAAI, 1996.

[BFH+98] C. Baral, L. Floriano, A. Hardesty, D. Morales, M. Nogueira, and T. Son. From Theory to Practice - The UTEP robot in the AAAI 96 and AAAI 97 robot contests. In *Proc. of Autonomous Agents 98 (to appear)*, 1998.

[BG93] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of 13th International Joint Conference on Artificial Intelligence, Chambery, France*, pages 866–871, 1993.

[BG97a] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1-3):85–117, May 1997.

[BG97b] C. Baral and A. Ghosh. BDI aspects of agents based on action theories. Technical report, Dept of Computer Science, University of Texas at El Paso, 1997.

[BGD+95] C. Boutilier, M. Goldszmidt, T. Deadn, S. Hanks, D. Heckerman, and R. Reiter, editors. *Extending Theories of Action: Formal Theory and Practical Applications.* Working notes of AAAI Spring Symposium, AAAI Press, 1995.

[BGP97] C. Baral, M. Gelfond, and A. Provetti. Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming*, 31(1-3):201–243, May 1997.

[BGPml] C. Baral, A. Gabaldon, and A. Provetti. Formalizing Narratives using nested circumscription. In *AAAI 96*, pages 652–657, 1996 (Extended version with proofs available at http://cs.utep.edu/chitta/chitta.html).

[BK96] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *AAAI 96*, pages 1215–1222, 1996.

[BKMS95] R. Bonasso, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In M. Woolridge, J. Muller, and M. Tambe, editors, *Intelligent Agents - II*, pages 187–202. Springer, 1995.

[Bra87] M.E. Bratman. *Intensions, Plans, and Practical Reason.* Harvard University Press, 1987.

[Bro86] R. Brooks. A robust layered control system for a mobile robot. *IEEE journal of robotics and automation*, pages 14–23, April 1986.

[Bro87] F. Brown, editor. *Proceedings of the 1987 worskshop on The Frame Problem in AI.* Morgan Kaufmann, CA, USA, 1987.

[Bro91a] R. Brooks. Elphants don't play chess. In P. Maes, editor, *Designing Autonomous Agents*, pages 3–16. MIT Press, 1991.

[Bro91b] R. Brooks. Intelligence without reason. In *Proc. of IJCAI 91*, pages 569–595, 1991.

[BS92] B. Burmeister and K. Sundermeyer. Cooperative problem solving guided by intentions and perception. *Decentralized AI*, 3, 1992.

[BS97] C. Baral and T. Son. Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. of International Logic Programming Symposium (ILPS 97)*, pages 387–401, 1997.

[CH90] P.R. Cohen and H.Levesque. Intention is choice with commitment. *Artificial Intelligence*, 43(3), 1990.

[DB90] M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing probability of goal satisfaction. In *Proc. of 8th National Conference on Artificial Intelligence (AAAI 90)*, pages 138–144, 1990.

[DBS94] M. Drummond, J. Bresina, and K. Swanson. Just-in-case scheduling. In *Proc. of the Twelth National Conference on Artificial Intelligence*, pages 1098–1104, 1994.

[DGLL97] G. De Giacomo, Y. Lesperance, and H. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *IJCAI 97*, pages 1221–1226, 1997.

[Dij72] E. W. Dijkstra. The humble programmer. *CACM*, 15:859–886, October 1972.

[DKKN95] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76:35–74, 1995.

[DRS98] G. DeGiacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of KR 98*, 1998.

[Dru86] M. Drummond. *Plan Nets: a formal representation of action and belief for auomatic Planning Systems.* PhD thesis, Dept of AI, U of Edinburgh, 1986.

[Dru89] M. Drummond. Situation control rules. In *KR 89*, pages 103–113, 1989.

[EC82] E. A. Emerson and E. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. In *Science of Computer programming, vol 2*, pages 241–266. 1982.

[EHW+92] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *KR 92*, pages 115–125, 1992.

[EJ89] E. A. Emerson and J.Srinivasan. Branching time temporal logic. In J.W. de Bakker, W. P. de Roever, and G. Roezenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 123–172. Springer-Verlag, Berlin, 1989.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of theoretical computer science: volume B*, pages 995–1072. MIT Press, 1990.

[ENS95] K. Erol, D. Nau, and V.S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1-2):75–88, 1995.

[Fir87] R. Firby. An investigation into reactive planning in complex domains. In *AAAI 87*, 1987.

[Fir92] R. Firby. Building symbolic prmitives with continuous control routines. In *Proc. of the first international conference on AI planning systems (AIPS 92)*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[Fis93] K. Fisher. The Rule-based Multi-Agent System MAGSY. In *Pre-Proc. of the CKBS'92*. Kleele University, 1993.

[FN71] R. Fikes and N. Nilson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[Gat92] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-word mobile robots. In *Proc. of AAAI 92*, pages 809–815. Morgan Kaufmann, 1992.

[GD96] J. Gustafsson and P. Doherty. Embracing occlusion in specifying the indirect effects of actions. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *Proc. of KRR*, pages 87–98, 1996.

[Geo94] M. Georgeff, editor. *Journal of Logic and Computation, Special issue on Action and Processes*, volume 4 (5). Oxford University Press, October 1994.

[GEW96] K. Golden, O. Etzioni, and D. Weld. Planning with execution and incomplete informations. Technical report, Dept of Computer Science, University of Washington, TR96-01-09, February 1996.

[Gin89] M. Ginsberg. Universal planning: An (almost) universally bad idea. *AI magazine*, pages 40–44, 1989.

[GK91] P. Godefroid and F. Kabanza. An efficient reactive planner for synthesizing reactive plans. In *AAAI 91*, pages 640–645, 1991.

[GL87] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *AAAI 87*, 1987.

[GL93] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[GL95] E. Giunchiglia and V. Lifschitz. Dependent fluents. In *Proc. of IJCAI 95*, pages 1964–1969, 95.

[GLR91] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In R. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 167–180. Kluwer Academic, Dordrecht, 1991.

[GW96] K. Golden and D. Weld. Representing sensing actions: the middle ground revisited. In *KR 96*, pages 174–185, 1996.

[Haa86] L. Haas. A syntactic theory of belief and action. *Artificial Intelligence*, 28:245–292, May 1986.

[HFA94] P. Hayes, K. Ford, and N. Agnew. On babies and bathwater: a cautionary tale. *AI Magazine*, 15(4):14–27, winter 1994.

[HM87] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[Hoa87] C. A. R. Hoare. An overview of some formal methods for program design. *IEEE Cpmputer*, pages 85–91, September 1987.

[JB95] P. Jonsson and C. Backstrom. Incremental planning. In M. Ghallab and A. Milani, editors, *New Trends in AI Planning: Proc. 3rd European workshop on planning.* IOS Press, 1995.

[JB96] P. Jonsson and C. Backstrom. On the size of reactive plans. In *Proc. of AAAI 96*, pages 1182–1187, 1996.

[Jen92] N.R Jennings. Towards a cooperation knowledge level for collaborative problem solving. In *Proc. of ECAI 92*, pages 224–228, 1992.

[JF93] J. Jones and A. Flynn. *Mobile Robots.* A. K. Peters, 1993.

[Joh97] W. Johnson, editor. *Proceedings of the 1st international conferenec on autonmous agents.* ACM press, 1997.

[Kar98] L. Karlsson. Anything can happen: on narratives and hypothetical reasoning. In G. A. Cohn, L. Schubert, and S.C. Shapiro, editors, *Proc. of the Sixth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, 1998.

[KBSD97] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence (to appear)*, 1997.

[KGR96] D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In Walter van de Velde and J. Perram, editors, *Agents breaking Away – 7th European workshop on modelling autonomous agents in a multi-agent world*, pages 56–71. 1996.

[KL94] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[KNH97] D. Kortenkamp, I. Nourbaksh, and D. Hinkle. The 1996 AAAI mobile robot competition and exhibition. *AI magazine*, pages 25–32, 1997.

[KOG92] K. Krebsbach, D. Olawsky, and M. Gini. An empirical study of sensing and defaulting in planning. In *First Conference of AI Planning Systems*, pages 136–144, 1992.

[Kow95] R. Kowalski. Using metalogic to reconcile reactive with rational agents. In K. Apt and F. Turini, editors, *Meta-logics and logic programming*, pages 227–242. MIT Press, 1995.

[KR91] L. Kaelbling and S. Rosenschein. Action and planning in embedded agents. In P. Maes, editor, *Designing Autonomous Agents*, pages 35–48. MIT Press, 1991.

[Lan89] A. Lansky. A perspective on multi-agent planning. Technical Report 474, SRI, Menlo Park, CA, 1989.

[Lev96] H. Levesque. What is planning in the presence of sensing? In *AAAI 96*, pages 1139–1146, 1996.

[LHM91] D. Lyons, A. Hendriks, and S. Mehta. Achieving robustness by casting planning as adaptation of a reactive system. In *IEEE conference on Robotics and Automation*, 1991.

[Lif97] V. Lifschitz, editor. *Special issue of the Journal of Logic Programming on Reasoning about actions and change*, volume 31(1-3), May 1997.

[Lin97] F. Lin. An ordering on goals for planning - formalizing control information in the situation calculus. *Annals of Math and AI*, 21(2-4):321–342, 1997.

[Lin95] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI 95*, pages 1985–1993, 95.

[LL95] Y. Lesperance and H. Levesque. Indexical knowledge and robot action - a logical account. *Artificial Intelligence*, 73(1-2):69–115, 1995.

[LLL+94] Y. Lesperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high level robot programming – a progress report. In *Working notes of the 1994 AAAI fall symposium on Control of the Physical World by Intelligent Systems, New Orleans, LA*, November 1994.

[LLL+95] Y. Lesperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. Foundations of a logical approach to agent programming. In M. Woolridge, J. Muller, and M. Tambe, editors, *Intelligent Agents - II*, pages 331–346. 1995.

[LP96] R. Li and L. Pereira. Knowledge-based situated agents among us: a preliminary report. In J. Muller, M. Woolridge, and N. Jennings, editors, *Intelligent Agents III - ECAI 96 ATAL workshop*, pages 375–390. 1996.

[LR94] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, October 1994.

[LRL+97] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84, April-June 1997.

[LS91] F. Lin and Y. Shoham. Provably correct theories of actions: preliminary report. In *Proc. of AAAI–91*, pages 349–354, 1991.

[LS92] F. Lin and Y. Shoham. Concurrent actions in the situation calculus. In *Proc. of AAAI–92*, pages 590–595, 1992.

[LS95] F. Lin and Y. Shoham. Provably correct theories of action. *Journal of the ACM*, 42(2):293–320, March 1995.

[LTM97] J. Lobo, S. Taylor, and G. Mendez. Adding knowledge to the action description language $\mathcal{A}$. In *AAAI 97*, pages 454–459, 1997.

[Mae91a] P. Maes, editor. *Designing Autonomous Agents.* MIT/Elsevier, 1991.

[Mae91b] P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1991.

[McC59] J. McCarthy. Programs with common sense. In *Proc. of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationery Office.

[McD90] D. McDermott. Planning reactive behavior: a progress report. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 450–458, 1990.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[Mit90] T. Mitchell. Becoming increasingly reactive. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 459–467, 1990.

[MLLB96] G. Mendez, J Llopis, J. Lobo, and C. Baral. Temporal logic and reasoning about actions. In *Common Sense 96*, 1996.

[Moo77] R. Moore. Reasoning about knowledge and action. In *IJCAI 77*, page 223, 1977.

[Moo79] R. Moore. *Reasoning about knowledge and action.* PhD thesis, MIT, Feb 1979.

[Moo85] R. Moore. A formal theory of knowledge and action. In J. Hobbs and R. Moore, editors, *Formal theories of the commonsense world.* Ablex, Norwood, NJ, 1985.

[MS91] C. Malcom and T. Smithers. Symbol grounding via a hybrid architecture in an autonomous assembly system. In P. Maes, editor, *Designing Autonomous Agents*, pages 123–144. MIT Press, 1991.

[MS94] R. Miller and M. Shanahan. Narratives in the situation calculus. *Journal of Logic and Computation*, 4(5):513–530, October 1994.

[MT96] Y. Moses and M. Tennenholtz. Off-line reasoning for on-line efficiency: knowledge bases. *Artificial Intelligence*, 83:229–239, 1996.

[MT95] N. McCain and M. Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI 95*, pages 1978–1984, 95.

[Mul96] J. Muller. *The design of intelligent agents - a layered approach.* Springer, 1996.

[Mus94] D. Musliner. Using abstraction and nondeterminism to plan reaction loops. In *AAAI 94*, pages 1036–1041, 1994.

[MWJ96] J. Muller, M. Woolridge, and N. Jennings, editors. *Intelligent Agents III - ECAI 96 ATAL workshop.* Springer, 1996.

[Nil85] N. Nilsson. Triangle tables: a proposal for a robot programming language. Technical Report 347, AI Center, SRI International, 1985.

[Nil94] N. Nilsson. Teleo-Reactive programs for agent control. *Journal of AI research*, pages 139–158, 1994.

[OWA91] C. Ozveren, A. Willsky, and P. Antsaklis. Stability and stabilizability of discrete event dynamic systems. *Journal of the ACM*, pages 730–752, vol 38, no 3, July 1991.

[Pin94] J. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, University of Toronto, Department of Computer Science, February 1994. KRR-TR-94-1.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM POPL 1989*, pages 179– 190, 1989.

[PR93] J. Pinto and R. Reiter. Temporal reasoning in logic programming: A case for the situation calculus. In *Proceedings of 10th International Conference in Logic Programming, Hungary*, pages 203–221, 1993.

[PS92] M. Peot and D. Smith. Conditional non-linear planning. In *First Conference of AI Planning Systems*, pages 189–197, 1992.

[Rei91] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[Rei96] R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In L. Aiello, J. Doyle, and S. Shapiro, editors, *KR 96*, pages 2–13, 1996.

[Rei98] R. Reiter. *Knowledge in action: logical foundation for describing and implementing dynamical systems.* 1998. manuscript.

[RG91] A. Rao and M. Georgeff. Modelling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *KR 89.* 1991.

[RG95] A. Rao and M. Georgeff. BDI agents : From theory to practice. In *Proc. of the first international conference on multiagent systems.* 1995.

[RK95] S. Rosenschein and L. Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 73(1-2):149–173, 1995.

[RLU90] P. Rosenbloom, S. Lee, and A. Unruh. Responding to impasses in memory- driven behavior: a framework for planning. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 181–191, 1990.

[RW87a] P. Ramadge and W. Wonham. Supervisory control of a class of discrete event process. *SIAM J. Cont. Optimization 25*, 1, pages 206-207, Jan 1987.

[RW87b] P. Ramadge and W. Wonham. Modular feedback logic for discrete event systems. *SIAM J. Cont. Optimization 25*, 5, pages 206-207, Sept 1987.

[San89] E. Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In N. S. Sridharan, editor, *Proc. of the 11th Int'l Joint Artificial Intelligence*, pages 894–899, 1989.

[San92] E. Sandewall. Features and fluents: A systemetic approach to the representation of knowledge about dynamical systems. Technical report, Institutionen for datavetenskap, Universitetet och Tekniska hogskolan i Linkoping, Sweeden, 1992.

[San93] E. Sandewall. The range of applicability of nonmonotonic logics for the inertia problem. In Rezena Bajcsy, editor, *Proc. of the 13th Int'l Joint Conference on Artificial Intelligence*, pages 738–743, 1993.

[San94] E. Sandewall. *Features and Fluents: The representation of knowledge about dynamical systems.* Oxford University Press, 1994.

[San96] E. Sandewall. Assessments of ramification methods that use static domain constraints. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *Proc. of KRR*, pages 89–110, 1996.

[San97] E. Sandewall. Logic-based modeling of goal directed behavior. *Linkoping Electronic Articles in Computer and Information Science (also in KR 98)*, 2(19 (http://www.ep.liu.se/ea/cis/1997/019/)), 1997.

[SBK+93] D. D. Steiner, A. Burt, M. Kolb, , and C. Lerin. The conceptual framework of MAI_2L. In *Pre-Proc. of MAAMAW'93*, 1993.

[Sch87] M. Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI 87*, pages 1039–1046, 1987.

[Sch89a] M. Schoppers. In defense of reaction plans as caches. *AI Magazine*, 10(4):51–60, 1989.

[Sch89b] M. Schoppers. *Representation and Automatic Synthesis of Reaction Plans.* PhD thesis, University of Illinois at Urbana-Champaign, 1989.

[Sch92a] J. Schlipf. Complexity and undecidability results in logic programming. In *Workshop on Structural Complexity and Recursion-theoretic methods in Logic Programming*, 1992.

[Sch92b] M. Schoppers. Building plans to monitor and exploit open-loop and closed-loop dynamics. In *First Conference of AI Planning Systems*, pages 204–213, 1992.

[Sch94] M. Schoppers. Estimating reaction plan size. In *Proc. of AAAI 94*, pages 1238–1244, 1994.

[Sch95] M. Schoppers. The use of dynamics in an intelligent controller for a space-faring rescue robot. *Artificial Intelligence*, 73:175–230, 1995.

[Sel94] B. Selman. Near-optimal plans, tractability and reactivity. In *Proc. of KR 94*, pages 521–529, 1994.

[Sha97] M. Shanahan. *Solving the frame problem: A mathematical investigation of the commonsense law of inertia.* MIT press, 1997.

[Sho86] Y. Shoham. Chronological ignorance: Time, nonmonotonicity, necessity and causal theories. In *Proc. of AAAI-86*, pages 389–393, 1986.

[Sin94] M. Singh. *Multiagent systems - a theoretical framework for intentions, know-how, and communications.* Springer-Verlag, 1994.

[SKR95] A. Saffioti, K. Konolige, and E. Ruspini. A multivalued logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.

[SL93] R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *AAAI 93*, pages 689–695, 1993.

[ST92] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proc. of AAAI 92*, pages 276–281. Morgan Kaufmann, 1992.

[Thi97] M. Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1-2):317–364, 1997.

[Tho95] S.R. Thomas. The PLACA agent programming language. *Lecture Notes of Artificial Intelligence*, 890:355–370, 1995.

[TS95] P. Traverso and L. Spalazzi. A logic for acting, sensing, and planning. In *IJCAI 95*, pages 1941–1947. 1995.

[TSG95] P. Traverso, L. Spalazzi, and F. Giunchiglia. Reasoning about acting, sensing, and failure handling: a logic for agents embedded in real world. In M. Woolridge, J. Muller, and M. Tambe, editors, *Intelligent Agents - II*, pages 65–78. 1995.

[vdVP96] Walter van de Velde and J. Perram, editors. *Agents breaking Away – 7th European workshop on modelling autonomous agents in a multi-agent world.* Springer, 1996.

[Wag96] G. Wagner. A logical and operational model of scalable knowledge- and perception-based agents. In Walter van de Velde and J. Perram, editors, *Agents breaking Away – 7th European workshop on modelling autonomous agents in a multi-agent world*, pages 26–41. 1996.

[WMT95a] M. Woolridge, J. Muller, and M. Tambe. Agent theories, architectures, and languages: a bibliography. In M. Woolridge, J. Muller, and M. Tambe, editors, *Intelligent Agents - II*, pages 408–432. 1995.

[WMT95b] M. Woolridge, J. Muller, and M. Tambe, editors. *Intelligent Agents II (IJCAI 95 ATAL workshop).* Springer, 1995.

[Woo94] M. Woolridge, editor. *Intelligent Agents I (ECAI 94 ATAL workshop).* Springer, 1994.