Relating theories of actions and reactive control

Chitta Baral and Tran Cao Son

Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968
{*chitta,tson*}*@cs.utep.edu*

Abstract

In this paper [1] we give a formal characterization of reactive control using action theories. In the process we formalize the notion of a reactive control program being correct with respect to a given goal, a set of initial states, and action theories about the agent/robot and about the exogenous actions. We give sufficiency conditions that guarantees correctness and use it to give an automatic method for constructing provenly correct control modules. We then extend our approach to action theories and control modules that have specialized sensing actions and that encode conditional plans. Finally, we relate our theory with the implementation of our mobile robot Diablo, which finished third and scored 285 out of 295 points in the AAAI 96 mobile robot contest.

[1] Note:
An alternative title of this paper could be 'Formal characterization and automatic construction of reactive control using action theories'.

Preprint submitted to Elsevier Science

Contents

1 Introduction

A theory of action allows us to *specify*[2] the relationship among fluents[3] associated with the world 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, then 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,Rei91,LS91,LS95] that incrementally consider various *specification aspects* such as:

[2] One of the goal is that the specification be elaboration tolerant [McC59,MH69]. But most theories of actions are normally tolerant to only certain elaborations.

[3] 'Fluents' are either physical objects of the world that may change their values, or mental objects that encode the mental state of the robot/agent. For example, 'loaded' is fluent which tells us if the gun is loaded or not.

actions with non-deterministic effects [Bar95,Pin94,KL94], concurrent actions [LS92,BG93,BG97a], narratives [MS94,PR93,BGP97,BGP96], actions with duration [MS94,Rei96], natural events [Rei96], ramifications and qualifications due to simple and causal constraints [LR94,KL94,Bar95,Lin97,Thi97,MT95],[Lin95,Bar95,GL95], sensing (or knowledge producing) actions [SL93,JTM97], 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] or a complex plan [LLL$^+$94,LRL$^+$97]. The books [Sha97,San94], the special issues [Geo94,Lif97], and the workshop proceedings [Wor95,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 [4] 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 to repeatedly execute (i), (ii) and (iii) [5] 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

[4] Throughout this paper we use the terms 'state' and 'situation' interchangeably. The intuitive difference is that a (complete) state is a snap-sort of the world and describes what is true or false in the world; while a situation describes the history of the world in terms of what actions have taken place since the initial situation. Every situation is mapped to a state that describes what is true in that situation, and there may be more than one situations which correspond to the same state.

[5] Step (iii) is slightly modified so that the agent does not execute the whole plan constructed in step (ii), rather it executes a small portion of the plan, and goes back to step (i) to take into account any changes to the world during that execution.

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, shuttle control agents, etc.

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 [Dru89] and [Nil94] for other domains.

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 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 to mind is: Is there some relationship between action theories and reactive control? To the best of our knowledge, the relationship between them is not well established.[6] *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[7] .

[6] Some attempts were made in [SKR95,KR91,Dru89]. We discuss them in Section 3.

[7] In the literature the term 'exogenous events' is mostly used instead of 'exogenous actions'. We use 'exogenous actions' to maintain the link with the phrase 'action

– We develop sufficiency conditions that guarantee the correctness 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 [8], 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 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*[9] 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*

theory' that we use throughout the paper.
[8] These conditions are analogous to the restriction on the class of languages for which we can construct a parsing table.
[9] In Section 4.1 we motivate and elaborate on the intuition behind choosing minimal cost as our criterion.

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 [10] –
> for commonly encountered situations.

In fact, in our opinion, this is has a lot of similarity with human behavior. A significant amount [11] 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 learnt [12] 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. [13] 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 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

[10] 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.

[11] 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.

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

[13] 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.

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 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 commonsense 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

We hope we have motivated and made clear the goal of this paper. In the reminder of this paper we formalize the intuitive ideas discussed in the previous sub-section, and work out several examples to elaborate our formalization.

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. One of the non-trivial examples that we introduce in this section is used as a running example in later sections.

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 our 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. We present several motivating examples, and briefly review the research on reasoning and planning with knowledge producing actions.

In section 6 we formalize the correctness of the extended control modules and in section 6 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. Finally, in section 12 we relate the methodology in this paper to the views of Dijkstra and Hoare on programming methodologies in general and we then conclude.

2 Simple Control Modules: preliminaries

In this paper we consider a robot [14] control architecture consisting of hierarchically defined reactive control modules. 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 approach is suggested in [Sch87,Nil94,Fir92,BKMS95]. While, Nilsson does not stipulate the numbers of levels, both Firby [Fir92] and Bonasso et al. use three levels.

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.

[14] Throughout the paper when we use the term 'robot' we also mean an 'agent' and vice-versa except when we say 'mobile robot'.

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.[15] 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 Some Examples of Control Modules

Consider a mobile robot navigating the building in Figure 1. Assuming that the robots 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.

[15] Note that the operational semantics of control modules differs from that of production systems in that in production systems of all the rules whose LHS matches successfully the RHS of only one of those rules is executed. This rule is chosen either non-deterministically or using some conflict resolution strategy. Besides, the actions in the rules of production systems are normally simple actions that change the truth value of a fluent. In regards to production systems, Waldinger and Stickel in [WS92] describe a logical methodology to formally prove properties of such systems.

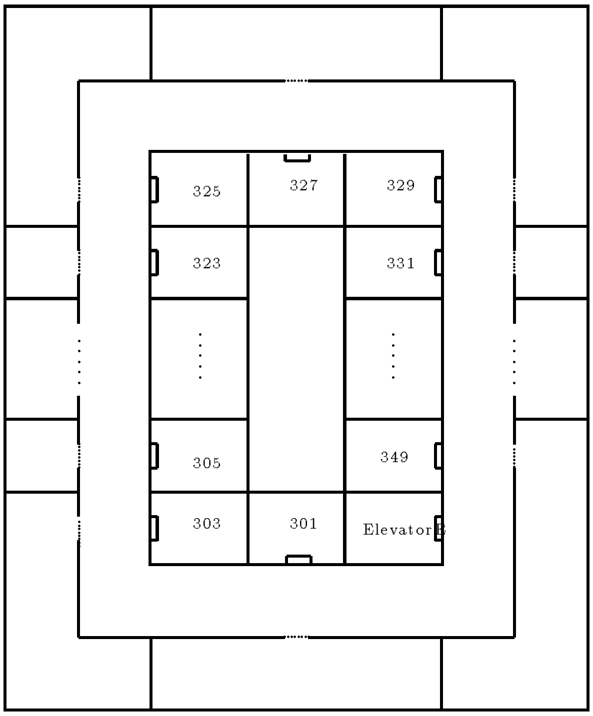

Fig. 1. Mobile robot in a simple office floor.

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 $(at\_room(301) \oplus \ldots \oplus at\_room(349) \oplus at\_elevator)$ [16]

It should be noted that for mobile robots the effects of actions and the constraints that we specify is from the robot's perspective[17] *. 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, 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 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

[16] The symbol $\oplus$ denotes ex-or.
[17] Lesperance and Levesque in [LL95] give a detailed logical account of knowledge from the robot's perspective – termed 'indexical knowledge'.

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 $false$. 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. 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 $del\_started$ changes the value of the mental fluent; 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 $go\_forward$, $turn\_rt\_90$ 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 specify 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.

2.3 Specifying actions and their effects

Surprisingly, a simple action theory without features such as being able to observe (as in $\mathcal{L}$ [BGP97]), or having narratives [MS94,PR93], is sufficient for formulating correctness of the control module discussed in the previous section.

This is because of the fact that the robot (with a reactive control) does not reason about its past. It just takes into account the current sensor values (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; it is prepared to sense again and proceed 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 $Res(a_n, Res(a_{n-1}, \ldots, Res(a_1, s) \ldots))$.[18] 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 - 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 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, and can reason about concurrent executions of sequences of actions is suitable for our purpose. Moreover, certain simplifications in the class of control modules we allow, such as the RHS being only one action, results in simplifying the required action theories. In this particular case, the simplification allows us to use the theory $\mathcal{A}_c$, which allows concurrent execution of a set of actions (not a set of sequence of actions). In most of the paper we will be using such a theory.

2.4 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. This module is not too small to be trivial, and hopefully is

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

not too big to make it difficult to analyze. 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 $(black\_door\_on\_rt \oplus w\_door\_on\_rt \oplus wall\_on\_rt \oplus corridor\_on\_rt)$
we know that in every state one and only one of the fluents $black\_door\_on\_rt$, $w\_door\_on\_rt$, $wall\_on\_rt$, 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 that 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 column $(j+1)^{th}$ of the first row 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 more 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 X' to denote the set $\{s' \mid s \in X\}$ and $s^*_{i,j,X}$ 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 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.

But, 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 **always** $(corridor\_on\_rt \Rightarrow \neg at(other))$ $s^*_{4,5,X}$ are not states in A.
- Due to **always** $(app\_corner \Rightarrow \neg at(other))$ the states $s^*_{i,5,app\_corner}$ are not states in A
- Due to **always** $(corridor\_on\_rt \Rightarrow \neg app\_corner)$ 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:

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

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 this section we formalize the notion of correctness of control modules. But, what is the intuitive meaning of being 'correct'? The first response would be that the control modules behaves according to specifications. But what are the specifications?

Let us first consider achievement control modules. *The initial specifications for achievement control modules are that when it is executed in a state from a given set of states, it always terminates in states where the achievement goal is satisfied.* The initial specification has two parameters: a set of states S, where the robot may initially start, and a goal G (which for the time being could be a formula of fluents), that the robot is trying to achieve. The correctness of a control module with respect to the initial specification can be intuitively described as follows: For each state s in S, we find the rule5 [19] (if any) whose LHS is satisfied, and consider the state s', that would be reached if the RHS of that rule is executed in s. (In our reasoning, we are not actually executing the RHS of the rule, rather we are doing 'hypothetical reasoning' about what state will be reached if the RHS of that rule is executed. For doing hypothetical reasoning, we need to use the action theory that describes effects of the robot's actions. *This adds another parameter to the notion of correctness – the action theory about the robot's actions.*) We then look for the rule whose RHS is satisfied by s' and consider the state that would be reached by executing this rule in s' and so on. We continue this until we reach a state which satisfies the LHS of a rule whose RHS is HALT, or we reach a state that does not satisfy the LHS of any rule. In the first case, if the final state satisfies the goal, we have convinced ourselves that if the control module is executed in state s then it will ultimately terminate in a state that satisfies the goal. (In the second case, the control module reaches a failure state.) By doing similar reasoning about all the states in S, we can convince ourselves about the correctness of the control module with respect to the set of initial states, the achievement goals, and the action theory about the robot's actions.

But this is not enough. So far we have not taken into account the occurances of exogenous actions. These actions are beyond the control of the robot and we do not know before hand when they will happen. Moreover, they are not part of the control module. Since we are interested in autonomous agents in a dynamic world, where actions other than the robot's own action may happen, we must take them into account.

We take exogenous actions into account by expanding the set S of initial states. We add to it states that the robot may be in while executing its control module

[19] For the sake of simplicity we assume here that there may be at most one rule whole LHS is satisfied.

and because of arbitrary occurances of several exogenous actions. These states are obtained by hypothetically reasoning about the effect of exogenous actions. [20] We then consider correctness with respect to all these added states. *Notice that we have now added another parameter to the notion of correctness – the action theory of exogenous actions.*

Based on these intuition, we now formally characterize the correctness of control modules with respect to a given set of initial states, an achievement goal, and action theories about the robot's actions and exogenous actions. But first, we start with a formal definition of the closure of a set of states with respect to a control module, and action theory about the exogenous actions. The definition of closure [21] formalizes the notion (discussed in the previous paragraph) of adding states to the initial state, so as to take into account the exogenous actions.

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:

(i) $S \subseteq S'$.

(ii) If $s \in S'$ and a is an action in A that can occur independent of the robot then $Res(a, s) \in S'$ [22]

(iii) If $s \in S'$ and there exist a rule in M whose LHS is satisfied by s and whose RHS is $a_1, \ldots, a_l$, then $[a_1, \ldots, a_l]s \in S'$ □

Proposition 3.1 For any set of states S, control module M and action theory

[20] This is a very important distinction between our approach and the approach of Brooks [Bro86]. Brooks suggests [Bro91a,Bro91b] getting rid of representation and reasoning and to test robots in the real world. Although, the robot can be tested with respect to the limited number of initial states, it is almost impossible to exhaustively create real world scenarios with exogenous actions occuring in different combinations and at different moments. One alternative would be to do simulation - but then we are not in the real world. Our approach of using formal reasoning presents another alternative. Later we discuss how our approach leads to automatic construction of control modules.

[21] The definition of closure we have next assumes that with respect to any state s, the LHS of at most one rule is satisfied by s. This assumption can be easily removed by considering concurrent execution of actions. For simplicity, and since all our examples in this paper are consistent with assumption, we use this assumption in most places.

[22] 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$\}$.

A, there is a unique set of states S' which is the closure of S w.r.t. M and A. $\square$

Proof: The complete proof is in Appendix A.
The proof is based on showing that closure of S w.r.t. M and A is the least fixpoint of the function $T_{M,A,S}$ defined below:

$$T_{M,A,S}(X) = S \cup X \cup R_{A,S}(X) \cup R_{M,S}(X)$$

where

$R_{A,S}(X) = \{Res(a, s) : s \in X$, a is a action in A that can occur independent of the robot$\}$, and

$R_{M,S}(X) = \{[RHS]s : s \in X$, **if** LHS **then** RHS is a rule in M, LHS satisfied in s and [RHS]s is defined $\}$ $\square$

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 $S = Closure(S, M, A)$. $\square$

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 assumption the set of initial states for the robot, which we will denote by S_1, in the notation described in section 2.4 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. (The detailed computation of the closure is given in Appendix B.)

Following are some exogenous actions that we may 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 $s^*_{4,j,\neg app\_corner}$.
- *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 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, $T_{M,A_1,S_1}(X) = S_1 \cup X \cup R_{M,S_1}(X)$.

(i) Since $R_{M,S_1}(\emptyset) = \emptyset$, $T^0_{M,A_1,S_1}(\emptyset) = S_1$.
(ii) 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,

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

(iii) 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,

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

(iv) 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,

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

(v) 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,

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

(vi) 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

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

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

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

The other closures are as follows:

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

$$Closure(S_1, M, A_3) = Closure(S_1, M, A_2) \cup$$
$$\{s_{3,j,\neg app\_corner} \mid j = 1, \ldots, 4\} \cup \{s_{4,j,\neg app\_corner} \mid j = 1, \ldots, 4\}$$

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

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 where there are no exogenous actions then it will reach the goal.

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, 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 $max_{s \in S}|\mathcal{U}_M(s)| = n$. $\Box$

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 it in the absence of exogenous actions it reaches the goal in less than n steps.

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

In this section we continue with our running example and 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.* 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 $max_{s \in S}|\mathcal{U}_M(s)| = n$. □

Consider the sufficiency 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_sirent **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 $\neg off\_siren$.

4 Sufficiency conditions for correctness of simple control modules

4.1 Sufficiency conditions for achievement control modules

Now that we have formalized correctness of control modules we would like 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, the sufficiency conditions may be used to automatically construct control modules. This is our goal in this section.

Our approach involves specifying two conditions on a control module: the *first* condition would make sure that each state in the closure is covered by at least one control rule, and the *second* condition would look at each control rule in the control module and make sure that it is 'individually sound'.

The first condition is formalized as follows.

Definition 4.1 *[Completeness]* An achievement control module M is said to be *complete* w.r.t. goal G and a set of states S, if for each s in S there exists at least one rule in M which is applicable in s. □

The second condition can be elaborated (using the notion of action leading to a goal by Kaelbling and Rosenschein) as that a simple control rule r is sound if for any state where r is applicable the *action* in the RHS of r '*leads to the goal.*'

We will have two different formalization of the notion 'leads to the goal':

- An action a *weakly leads* to a goal from a state s if there exists a minimal plan from s which achieves the goal and has a as its first action.
- 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.

Intuitively, we would like to show that if a control module satisfies the soundness and completeness conditions then it is correct.

Before we formally define the soundness condition we would like to argue why the 'weakly leading' condition, although intuitive, is not strong enough to guarantee correctness.

Consider the following control module to go to the elevator.

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 'weakly leads' to the goal. To verify that, 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 $Go\_anticlockwise\_1room$ 'weakly leads' to the goal from the state $\{at\_room(323)\}$. Similarly, the action $Go\_clockwise\_1room$ 'weakly leads' to the goal from the state $\{at\_room(321)\}$.

But, to our initial surprise, 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.

This demonstrates the weakness of the 'weakly leading' condition. We now formally define the soundness condition based on the 'action leading to goal' notion.

Definition 4.2 *[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). □

One of the 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.

Another cost assignment, which we prefer, is to define, $cost(a, s)$, the cost of action a in situation s, to be the number of changes in truth value of the fluents between the states s, and $res(a, s)$. The reason we prefer this definition is that, in general we would like to have as little change as possible during the execution, and we count the changes when a fluent starts out true, becomes false during execution and returns to true at the end. An example which makes the point (with some exaggeration) is comparing plans that achieve the goal of destroying a paper document. A single action of blowing up the room would do it. Another plan would be to take the paper to the department office that has a shredder, and put the paper in the shredder. The later plan, although with two actions instead of one, is definitely preferable than the former, as it does less change to the world.

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 4.3 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. $\square$

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. But first we will state two lemmas that lead to the main theorem.

Lemma 1 If M is sound and complete w.r.t. a goal G then for every $s \in S$ if the rule **if** LHS **then** RHS is applicable in S and P is a minimal cost plan that achieves G from s such that $P = RHS \circ Q$ then Q is a minimal cost plan that achieves G from $[RHS]s$.

Lemma 2 If M is sequential w.r.t. a set of states S, and is sound and complete w.r.t. a goal G and S then $\mathcal{U}_M(s)$ is finite for $s \in S$.

Theorem 1 Let A be an action theory. Consider a pair (M, S), where S is a set of complete states, and 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 and complete w.r.t. G and S then M achieves G from S w.r.t. A. $\square$

Proof: The detailed proof is 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 $\{\{at\_elevator\}, \emptyset\}$, when the action theory only consist of the action $Go\_clockwise\_1room$. We can then use Theorem 1 to verify its correctness.
  When we add the action $Go\_anticlockwise\_1room$, it is no longer sound, but only weakly sound. We can no longer use Theorem 1 to verify its correctness.
- The control module $Goto\_elevator\_2$ is sound and complete with respect to the set of states $\{\{at\_elevator\}, \emptyset\}$, when the action theory consists of actions $Go\_clockwise\_1room$ and $Go\_anticlockwise\_1room$. Therefore, we can use Theorem 1 to verify its correctness.
- The control module $Go\_clockwise\_1room$ is sound and complete with respect to the set of all states $\{\{at\_elevator\}, \emptyset\}$, and the action theory in Section 2.2. Therefore, we can also use Theorem 1 to verify its correctness.
- The soundness and completeness conditions and Theorem 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 verify the correctness of some control modules. *But, its more important significance is that it can be used to automatically construct control modules.* In this section we give two algorithms to automatically construct achievement control modules. The first algorithm uses the soundness and completeness condition and Theorem 1 to construct a control module that achieves a goal G from a set of states S, assuming that G is achievable from all states in $Closure(S, A, A)$, where A is the action theory. Intuitively, this algorithm iteratively updates the closure and adds a control rule that satisfies the soundness criteria for each state in that is already in the closure.

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 $M = \{$ **if** G **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 $M = M \cup \{$ **if** s **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 [23]. □

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}. (A formal proof of this is similar to the corresponding part of the proof of Proposition 4.2.) 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 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 an 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"

[23] For example, the rules **if** $f, \neg g$ **then** a and **if** f, g **then** a can be merged to the single rule **if** f **then** a. More efficient merging algorithms can be obtained using techniques in digital circuit design where truth tables are optimally realized by considering fluents as input variables to the circuits and actions as output variables of the circuit. The idea of realizing control modules as circuits was earlier discussed in [Nil94].

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 least cost plan in Step 2.2, this can be used as a choice point to backtrack, to look for other ways to have $S_{in} = S_{out}$.

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 $M = \{$ **if** G **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 $M = M \cup \{$ **if** $[a_1, \ldots, a_j]s$ **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.

4.3 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 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

5.1 Motivation

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. *But we did not consider special sensing actions or knowledge producing actions in their control modules.*

Consider the specification of the office navigation part of the AAAI 96 robot contest. In it, the robot initially does not know if a particular conference room is occupied or not. Depending on whether it is occupied or not the robot was required to do different things. One way to approach this problem is for the robot to navigate to the conference room and then do a special sensing action to determine if the conference room is occupied or not. The sensing that the robot does in the sense and act cycle during its navigation to the conference room is referred to as *regular sensing*. On the other hand, the sensing that it does in the acting phase to determine if the conference room is occupied or not, is referred to as *special sensing*. We make this distinction because we do not want the robot to do all kinds of sensing all the time. In particular, some sensing actions are expensive (or time consuming), particularly ones involving analyzing a pattern or a picture, and should be done only when they are necessary. Such special sensing actions 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 knowledge producing actions [SL93]. (A plan based on such a theory could be a conditional plan [Lev96] and may achieve knowledge goals [GEW96].) 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.

5.2 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.

5.2.1 An example of a Control Module with sensing actions

We have a robot which can perform the actions: *go_to_fridge, look_at_fridge, open_fridge, look_into_fridge, get_milk_from_fridge*, and $get\_milk\_from\_store$. Intuitively, if the robot performs the action go_to_fridge it will be *near the fridge*. If it then performs the action $look\_at\_fridge$ then it can find out if the *fridge is open* or not. By performing the action $open\_fridge$, the fluent $fridge\_open$ becomes true. Let us now consider a control module which a mobile robot can execute to get milk, either from the fridge or from the store.

Example 5.1 Control Module – Get_Milk
if $\neg has\_milk,\ u(fridge\_open), \neg near\_fridge$ **then** go_to_fridge
if $\neg has\_milk,\ u(fridge\_open), near\_fridge$ **then** $look\_at\_fridge$
if $\neg has\_milk,\ \neg fridge\_open, \neg near\_fridge$ **then** go_to_fridge
if $\neg has\_milk,\ \neg fridge\_open, near\_fridge$ **then** $open\_fridge$
if $\neg has\_milk,\ fridge\_open,\ u(milk\_in\_fridge), near\_fridge$
  then $look\_into\_fridge$
if $\neg has\_milk,\ fridge\_open,\ u(milk\_in\_fridge), \neg near\_fridge$
  then go_to_fridge
if $\neg has\_milk,\ milk\_in\_fridge,\ fridge\_open, \neg near\_fridge$ **then** go_to_fridge
if $\neg has\_milk,\ milk\_in\_fridge,\ fridge\_open, near\_fridge$ **then** $get\_milk\_from\_fridge$
if $\neg has\_milk,\ \neg milk\_in\_fridge$ **then** $get\_milk\_from\_store$
if has_milk **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.

5.3 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 2.3, 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. 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. 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 action, 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 Get_Milk

go_to_fridge **causes** $near\_fridge$
$look\_at\_fridge$ **determines** $fridge\_open$ **if** $near\_fridge$
$open\_fridge$ **causes** $fridge\_open$ **if** $near\_fridge$
$look\_into\_fridge$ **determines** $milk\_in\_fridege$ **if** $fridge\_open, near\_fridge$
$get\_milk\_from\_fridge$ **causes** has_milk **if** $milke\_in\_fridge, fridge\_open, near\_fridge$
$get\_milk\_from\_store$ **causes** has_milk

In the above rules, $look\_at\_fridge$ and $look\_into\_fridge$ are sensing actions while the others 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 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 Get_Milk, with respect to the incomplete state, $(\langle\emptyset, \{has\_milk\}\rangle)$, where the robot is unaware of the truth value of any fluents, intuitively, results in the following plan.

Conditional Plan - $Plan\_Get\_Milk$
go_to_fridge;
$look\_at\_fridge$;

Case
$\neg fridge\_open \rightarrow open\_fridge$;
  $look\_into\_fridge$;
  Case
  $milk\_in\_fridge \rightarrow get\_milk\_from\_fridge$
  $\neg milk\_in\_fridge \rightarrow get\_milk\_from\_store$
  Endcase
$fridge\_open \rightarrow look\_into\_fridge$;
  Case
  $milk\_in\_fridge \rightarrow get\_milk\_from\_fridge$
  $\neg milk\_in\_fridge \rightarrow get\_milk\_from\_store$
  Endcase
Endcase

We refer to such a plan as a conditional plan. 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.

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:

(i) states to encode what the robot knows,

(ii) and allow reasoning about sensing actions and conditional plans.

We are aware of three such theories [SL93,JTM97,BS97]. In [SL93], states are Kripke models, while in [JTM97] 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 [SL93] and [JTM97] 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 follows that encodes the above ideas, it can be shown that $\models holds(has\_milk, [P]\langle\emptyset, \emptyset\rangle)$ where P stands for the conditional plan $Plan\_Get\_Milk$ specified in the previous example.

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 [24] of σ then σ' is in $Closure(S, M, A)$.

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

– 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 and whose RHS is $a_1, \ldots, a_l$, $(l > 1)$, then $\Phi(a_1; \ldots; a_l, s) \subseteq Closure(S, M, A)$.

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. $\Box$

Proof: Similar to the proof of Proposition 3.1.

We refer 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 $S = Closure(S, M, A)$. $\Box$

We will now formally characterize the effect of executing control modules with explicit 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** $a_1, \ldots, a_n$ 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
  $\mathcal{U}_M(s) = \alpha;$
    Case
    $s_1 \rightarrow \mathcal{U}_M(s_1)$
    $\vdots$
    $s_n \rightarrow \mathcal{U}_M(s_n)$
    Endcase

· 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 is said to achieve goal G from a set of states S, if for all s in S $\mathcal{U}_M(s)$ 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, which 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 4.1. Although, the intuitive idea behind the soundness condition here will be the same as before in section 4.1, we need to realize 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 the definition of a minimal conditional plan.

Definition 7.1 (Sub-plans of conditional plans and minimal conditional plans) Given a conditional plan C. A plan C' which is not identical to C is called a sub-plan of C if

(i) C' is the empty plan [], or
(ii) 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
(iii) C is a case plan

C = 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

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

$$
\begin{aligned}
C' = \text{case}\quad & \\
& p_{j_1,1}, \ldots, p_{j_1,m_{j_1}} \rightarrow c'_{j_1} \\
& \vdots \\
& p_{j_k,1}, \ldots, p_{j_k,m_{j_k}} \rightarrow c'_{j_k} \\
& \text{endcase}
\end{aligned}
$$

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},

(iv) 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 $1 \leq i \leq n$.

□

The following property of minimal conditional plans will later be useful in proving our main result of this section.

Lemma 3 Let C be a minimal conditional plan w.r.t (G, s). Then, C is empty or C starts with an action. □

Proof: In Appendix D.

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 7.2 (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

(i) 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

(ii) if none of the $p_{i,1}, \ldots, p_{i,n_i}$ are true w.r.t. s then $Unfold(C, s) = []$.

Definition 7.3 (Ordering between Conditional Plans) Let S be a set of complete states, and C_1 and C_2 be two conditional plans which achieves G from a possibly incomplete state s. We say

(i) $C_1 \leq_{S,s} C_2$ if for all s' in S which is a complete extension of s, $cost(Unfold(C_1, s')) \leq$

$cost(Unfold(C_2, s'))$.

(ii) $C_1 <_{S,s} C_2$ 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 7.4 (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 minimal 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 7.5 *[Completeness]* An achievement control module M is said to be *complete* w.r.t. goal G and 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 7.6 [Special prefix of conditional plan] Given a conditional plan C, the special prefix of C, denoted by $pref(C)$, is defined inductively as follows:

(i) If C is the empty plan [] then $pref(C) = []$.
(ii) If C is an action a then $pref(C) = a$.
(iii) If $C = a_1; \ldots; a_n; P$ where P is a case plan then $pref(C) = a_1; \ldots; a_n$.
(iv) If $C = C_1; C_2$ and C_1 does not contain a case plan then $pref(C) = pref(C_1); pref(C_2)$.
(v) 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 sue 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 actions 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.

Definition 7.7 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, 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 7.8 *[Soundness]*

(i) 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 knowledge-preserving[25] plan C w.r.t (G, s) (w.r.t $\leq_{S,s}$) 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 $pref(C)$.
(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). □

The following lemmas lead us to the main result of this section. The next lemma 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 lemma 5 states that the special prefix of plans with case plans always have a sensing action. These two lemmas are used in proving the main theorem of this section that states that control modules satisfying the sufficiency conditions are correct. The lemmas are used particularly in showing that the unfolding of a sound and complete control module results in a finite conditional plan.

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

Lemma 4 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 $f \in T' \cup F'$. $\Box$

Proof: Follows directly from the definition of knowledge-preserving, and the fact that a is knowledge-preserving in s. $\Box$

Lemma 5 Let S be a set of states and C be a minimal conditional plan that achieves G from s. If C contains a case plan and $pref(P)$ is knowledge-preserving in s then

(i) $pref(P)$ contains a sensing action and
(ii) s is an incomplete state.

$\Box$

Proof: In Appendix D.

Theorem 2 Consider a pair (M, S), where S is a set of (possibly incomplete) states, and 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 and complete w.r.t. G and S then M achieves G from S. $\Box$

Proof: In Appendix D.

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

We now give the sketch of an algorithm that uses Theorem 2 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 $M = \{$ **if** G **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*[26] P from s that achieves

[26] We will need a modification of the planners in [EHW+92] for constructing minimal conditional plans.

G. Let a be the first action of P.
2.3 If P is a non-null plan and $s = \langle\{a_1, \ldots, a_n\}, \{b_1, \ldots, b_m\}$ is incomplete then
$M = M \cup \{$ **if** $a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m, u(c_1), \ldots, u(c_k)$ **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 $M = M \cup \{$ **if** s **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. $\square$

Proposition 7.1 Given a goal G, a set of states S and an action theory A if G is achievable from all states in $Closure(S, A, A)$ through knowledge preserving plans, then the above algorithm generates a control module M such that M achieves G from $Closure(S, M, A)$. $\square$

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 2 the algorithm computes a control module that achieves the goal G from the set of states $Closure(S, M, A)$. $\square$

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 state by executing the given control module, or through occurances 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 their, 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[27] 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.* In this section we relate the theory described in the previous sections to the design of our contest entry, Diablo. In the process we present a top-down description of the control architecture of our robot. *We also point out future research directions suggested by our robot control implementation.*

9.1 The overall goal

In the AAAI 96 robot navigation contest [KNH97] robots were required to achieve a particular navigational task around a office like environment, given the topological map of the environment. In particular, the robot was required[28] 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 empty, 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, without changing the availability status of the conference rooms, and the robot was required to be back at the director's office.

[27] 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.

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

9.2 The top-level module and its correctness

Our top-level module was a control module of the kind described in Section 5.2. Its reactive structure was geared towards gracefully recovering 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, in particular, 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.

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

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 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.*

9.3 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.

9.4 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.

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

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

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

$$detect\_corridor \textbf{ causes } at(X), \neg at(Y) \textbf{ if } at(Y), type(Y, junction), type(X, corridor), heading(dir), adj(X, Y, dir)$$
$$turn\_180 \textbf{ causes } heading(west), \neg heading(east) \textbf{ if } heading(east)$$

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

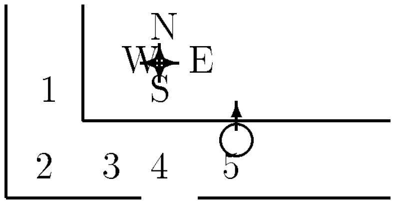

Fig. 2. A sample map.

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.

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 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.*

9.5 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

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.

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 9.5 list the skills, and basic actions of our robots, respectively, and their intuitive function.

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.*

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

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, resemble universal plans [Sch87] 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 as defined and used in the literature [Sch87,Gin89,JB96] 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 is an important contribution of our approach.*

In contrast to the approach in universal plans, which is criticized for its intractability [Gin89,JB96] [29] , 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:

- 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 gurantedly construct correct control modules, and we also experimentally validate our approach in the domain of a mobile robot in an

[29] 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. Another formal treatment of universal plans is done in [Sel94].

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. Moreover, we earlier argued that it is more intuitive.

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. But, 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, some of the additional contributions of this paper with respect to earlier research on universal plans and situation control modules are:

- In our definitions of correctness, our sufficiency conditions, and our control module construction algorithms, we do not use any specific action theory, rather we use an abstract 'entailment relation'. Because of this, our formulation need not change with more sophisticated action theories. In fact, our example on going to the elevator, used domain constraints. In contrast most formulations of Universal plan only use STRIPS.

– We present a formal treatment of 'maintainance goals' in presence of exogenous actions. Most formulation of universal plans only consider achievement goals.
– We consider special sensing actions. To our knowledge, no earlier formulation of universal plans or situation control modules consider such actions, and the encoding of conditional plans through control rules.

10.2 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, particularly lately, most of which are not 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.

10.2.1 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 occurances 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 co-operating 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].

10.2.2 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], 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. Using an extended version of situation calculus, the authors formulate correctness (with respect to a given goal) and termination of GOLOG programs. They also implement an interpreter of GOLOG in PROLOG. The aim of the interpreter is to take a given GOLOG program, verify its termination properties and in the process instantiate variables, and then execute the instantiated GOLOG program. *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[30]. *GOLOG allows non-deterministic actions, and recursive procedures while we do not. On the other hand GOLOG does not consider exogenous actions, and is not suitable for an agent in a dynamic 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.* Like the early approaches based on STRIPS it is only suitable for a robot in a static world. All the above criticisms are also applicable to CONGOLOG and $\mathcal{R}$, both of which allow sensing actions and the former allows multiple co-operating agents. Finally, GOLOG,

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

CONGOLOG and $\mathcal{R}$ are not concerned with automatic construction of robot programs, which is one of our main concerns.

Many theories of actions (including GOLOG and CONGOLOG) do not allow specification of observations and execution of actions. Hence, they are not able to adequately capture dynamic worlds. Recently, some action theories have been proposed [PR93,BGP97] 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 reactive 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, to the best of our knowledge, all earlier work on universal plans and control programs did not consider overcoming incompleteness using actions. In our formulation where we use sensing actions, we greatly benefited from [SL93,EHW^{+}92,Lev96,GEW96]. The main worry 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,Sch92]. An initial logical account of knowledge producing actions was given 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 Future Directions

In this section we discuss several issues related to the research reported here that need to be addressed further.

11.1 Hierarchical control modules as a reactive robot execution language

In the previous section we briefly discussed the robot execution languages GOLOG, CONGOLOG, and $\mathcal{R}$ in [LLL+94,LRL+97,LLL+95,Lev96] and pointed out that these languages are used in a non-reactive manner – as they are first verified with respect to termination properties, which results in some instantiations, and then executed.

On the other hand our approach specifying the control of a robot through hierarchical control modules can be considered as a reactive robot execution language. In fact syntactically, it could be a GOLOG program. The difference being that in our approach verification is done off-line (not just before the execution) and instantiations are done on-line (not during verification). *This is a preliminary analysis of our language as a reactive robot execution language. We would like to further analyze our language – particularly its expressive power, in line of the discussion in the last section of [Lev96]. We would also like to compare our language with the languages in [LLL+94,LRL+97,Lev96] in greater detail and consider extensions such as adding the notion of an interrupt.*

We will now formulate an example from [Lev96] in our language and point some additional differences between our approach and $\mathcal{R}$, and also list some future directions. The paper [Lev96] contains the following plan in $\mathcal{R}$ to make an omelette.

/*Assuming the bowl and the saucer are empty initially */
until there are three eggs in the bowl **do**
until there is an egg in the saucer **do**
break an egg into the saucer;
smell the saucer;
if the saucer has a bad egg
then discard its contents
end until;
transfer the contents of the saucer to the bowl
end until

Now we will try to encode the above plan using control modules as defined in this paper. First we will have a control module **Make_Omelette** with the following control rules:

- **if** $3eggs\_in\_bowl$ **then** $cook\_eggs, HALT$

- **if** $\neg 3eggs\_in\_bowl$ **then** $get\_one\_more\_egg\_in\_bowl$

The above control module has two actions, $cook\_eggs$ and $get\_one\_more\_egg\_in\_bowl$. The first action causes omelette to be ready when there are three eggs in the bowl. The second action adds one more good egg to the bowl. At this level of abstraction, both these actions are deterministic, and our control module is superior to the plan from [Lev96] in the sense that it does not assume that initially there are no eggs in the bowl. Our module can take advantage of opportunities and mischieves where the cooks helper may add additional eggs (but never more than 3 in the bowl) or spill the bowl of eggs. On the other hand the above control module does not specify how exactly the robot is going to execute the action $get\_one\_more\_egg\_in\_bowl$. As before, we will now give a control module that further details the action **get_one_more_egg_in_bowl**.

- **if** $\neg good\_egg\_in\_saucer$ **then** $get\_good\_egg\_in\_saucer$

- **if** $good\_egg\_in\_saucer$ **then** $transfer\_to\_bowl, HALT$

The non-unit action **get_good_egg_in_saucer** can be further expressed by the following control module.

- **if** $saucer\_empty$ **then** $break\_egg\_into\_saucer$

- **if** $\neg saucer\_empty, u(smell)$ **then** $smell$

- **if** $\neg saucer\_empty, bad\_smell$ **then** $discard\_egg$

- **if** $\neg saucer\_empty, \neg bad\_smell$ **then** $HALT$

The control module **get_one_more_egg_in_bowl** has good reactive properties. It can take advantage of the opportunity provided when its helper puts a good egg in the saucer, and it can overcome the helpers clumsiness when he/she spills the good egg in the saucer.

But, what if while the robot is executing **get_one_more_egg_in_bowl**, the helper adds good eggs to the bowl (not the saucer) such that there are three eggs in the bowl. The correct behavior of the robot would be to stop executing **get_one_more_egg_in_bowl** and return back to the **Make_omelette** module. That does not always happen when we have hierarchy of control modules. Although, it is easier to come up with hierarchy of modules, we loose reactiveness in the process. What is needed to restore reactivity is to *construct a single control module from a hierarchy of control modules*. In the above example the control modules **Make_omelette** and **get_one_more_egg_in_bowl** can be merged to the following module.

- **if** $3eggs\_in\_bowl$ **then** $cook\_eggs, HALT$
- **if** $\neg 3eggs\_in\_bowl, \neg good\_egg\_in\_saucer$ **then** $get\_good\_egg\_in\_saucer$
- **if** $\neg 3eggs\_in\_bowl, good\_egg\_in\_saucer$ **then** $transfer\_to\_bowl$

The above module can be further merged with the module **get_good_egg_in_saucer** to create the following module:

- **if** $3eggs\_in\_bowl$ **then** $cook\_eggs, HALT$
- **if** $\neg 3eggs\_in\_bowl, good\_egg\_in\_saucer$ **then** $transfer\_to\_bowl$
- **if** $saucer\_empty, \neg 3eggs\_in\_bowl, \neg good\_egg\_in\_saucer$ **then** $break\_egg\_into\_saucer$
- **if** $\neg saucer\_empty, \neg 3eggs\_in\_bowl, \neg good\_egg\_in\_saucer, u(smell)$ **then** $smell$
- **if** $\neg saucer\_empty, bad\_smell, \neg 3eggs\_in\_bowl, \neg good\_egg\_in\_saucer$ **then** $discard\_egg$

One of our main future goal is to develop conditions when a hierarchy of modules can be merged to a single module and to come up with algorithms to do that.

Now, let us go back to the module **get_good_egg_in_saucer**. So far we have only considered actions with deterministic effects. But the action, *break_egg_into_saucer* has non-deterministic effects. Its effect can be either *bad_smell* or $\neg bad\_smell$. *We need to extend our formalism to take into account such actions with non-deterministic effects and be able to prove the correctness of a plan in their presence.* Note that such plans, may have loops. The module **get_good_egg_in_saucer** encodes such a loop. But since our definition of a plan does not allow loops, we will not be able to unfold that module and have a finitely described plan. *What we need to do is to extend our definition of a plan to allow loops, and extend our definition of unfold such that the unfolding of the above module is a finitely described extended plan (possibly with loops). Also, we need to extend our action theory to allow non-deterministic actions and have an entailment from which we can entail correctness of extended plans.*

11.1.1 Goals in LHS of control rules

Our formalization in this paper assumes that the goal of the agent does not change during the execution. One of our future goal is to remove this assumption and allow the agent to change its goal which may be conveyed to the robot through sensory inputs or through a keyboard.. A control module for such a case would have control rules of the form:

if $p_1, \ldots, p_n$ **and goal** $q_1, \ldots, q_m$ **then** $a_1, \ldots, a_k$

where, $p_1, \ldots, p_n$ are fluents that correspond to the environment and the state of the robot, while $q_1, \ldots, q_m$ are fluents that correspond to the goal of the robot.

We plan to investigate if such an extension is really necessary, or if it can be stated using simple control modules (without the q's). If the former, we would like to extend our formalization in this paper to such extended control modules.

11.2 Failure and recovery

So far we have not discussed the issue of an action executed by our agent failing to have its desired effect.

If a particular occurances of an action *arbitrarily* fails to have its desired effect, our agent will be able to overcome it by treating it as an exogenous action. If after the failure, the world is in an accounted-for state; I.e., our control module has a rule to deal with this state, then it will overcome the failure. On the other hand if after the failure, the world is in an unaccounted-for state, then, as in Section 8, our agent will do on-line deliberation to reach one of the accounted for states.

The key qualification in the last paragraph is the word 'arbitrarily'. When we say an occurances of an action arbitrarily fails, we mean that it is very likely that when the same action is executed again it may not fail.

On the other hand when an action consistently fails, our approach above may not work. For example, suppose an action which is supposed to change the world in a particular way, fails in such a way that it does not change the world at all. Since the world is in the same state, the control module will execute the same action again, and since this action consistently fails, the control module will keep on executing the same action and the agent will never achieve its goal. Of course we can argue that this is similar to repeated harassment by exogenous actions, and no agent can achieve its goal if it is continually harassed. *But in future work we would like to explore ways to recognize such failures and have the agent pursue alternate means – without executing the action that failed consistently, to reach its goal.* Some research along this line is reported in [TSG95,TS95].

11.3 Complex Goals

In sections 3 and 5 achievement and maintainance goals are formulas of fluents which characterizes states that the robot intends to achieve or maintain. As mentioned in Section 9.2, more generalized goals characterize the trajectory that the world goes through instead of just characterizing a set of goal states. Although, our formulation of correctness can be easily generalized to such goals, our algorithms that construct control modules do not seem to lead to a simple extension that would be appropriate for such extended goals. We intend to look for alternative algorithms and would also like to explore if complex temporal goals, or particular subclasses of it can be decomposed to achievement and maintainance non-temporal goals. That would be useful as we can then use our present algorithms.

11.4 Co-operative control with multiple agents

Our formalization in this paper is about the control of a single agent in a dynamic world with exogenous actions. We would like to extend it to multiple co-operative agents, and formalize the correctness of a set of control modules corresponding to a set of co-operative agents. Although, CONGOLOG [LLL$^+$95] is such a formalism we would like to further allow exogenous actions and also look for automatic control module generation algorithms.

11.5 Correctness and automatic construction of fuzzy control modules

For many tasks, such as the low level control of a mobile robot, where sensor and actuator uncertainty is critical, fuzzy control rules [SKR95], or control rules based on intervals [KK96] is more appropriate. In [SKR95], Saffioti et al formulate the correctness of such 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 search for algorithms that will generate control modules based on multi-valued logic.

11.6 Implementation Issues

In section 9, we discussed the implementation of our AAAI 96 robot contest entry. We would like to implement many of the concepts discussed in this paper, starting with an automatic verifier of control modules, an interpreter for the language of hierarchical control modules, and a compiler that would *efficiently* generate control modules for classes of action theories. We hope to be able to use the conditional planner (with appropriate modifications) developed in [GW96].

12 Conclusion

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, to-

gether 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.

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.

[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.

[BGP96] C. Baral, A. Gabaldon, and A. Provetti. Formalizing Narratives using nested circumscription. In *AAAI 96*, 1996.

[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.

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

[BKMS95] R. Bonasso, D. Kortencamp, 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. Technical report, Dept of Computer Science, University of Texas at El Paso, 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.

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

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

[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.

[ENS95] K. Erol, D. Nau, and V.S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *AI Journal*, 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.

[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, 19889.

[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.

[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.

[JTM97] Lobo. J., S. Taylor, and G. Mendez. Adding knowledge to the action description language $\mathcal{A}$. In *AAAI 97*, 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.

[KK96] R. Kearfott and V. Kreinovich. *Applications of Interval Computations.* Kluwer, Dordrecht, 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*, 1996.

[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 (to appear)*, 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 (to appear), 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*, May 1997.

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

[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.

[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.

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

[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.

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

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

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

[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 teh situation calculus. In L. Aiello, J. Doyle, and S. Shapiro, editors, *KR 96*, pages 2–13, 1996.

[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.

[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.

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

[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.

[Sch92] J. Schlipf. Complexity and undecidability results in logic programming. In *Workshop on Structural Complexity and Recursion-theoretic methods in Logic Programming*, 1992.

[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 (to appear), 1997.

[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 logicfor 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.

[Wor95] Working notes of AAAI Spring Symposium. *Extending Theories of Action: Formal Theory and Practical Applications.* AAAI Press, 1995.

[WS92] R. Waldinger and M. Stickel. Proving properties of rule based systems. *International journal of software engineering and knowledge engineering*, 2(1):121–124, 1992.

Appendix A – Existence of unique closure

In the following we will prove that given S, M, and A there exists a unique set of states which is the closure of S w.r.t. M and A. We first prove two lemmas which lead to this proposition.

Lemma 6 *For any set of states S, control module M and action theory A, the function $T_{M,A}$ which is defined by,*

$$T_{M,A,S}(X) = S \cup X \cup R_{A,S}(X) \cup R_{M,S}(X)$$

where,

$R_{A,S}(X) = \{Res(a,s) : s \in X,$ *a is a action in A that can occur independent of the robot*$\}$, *and*

$R_{M,S}(X) = \{[RHS]s : s \in X,$ **if** *LHS* **then** *RHS is a rule in M, LHS is satisfied in s and [RHS]s is defined* $\}$
is monotonic.

Proof: It is easy to see that if $S_1 \subseteq S_2$ then
$R_{A,S}(S_1) \subseteq R_{A,S}(S_2)$ and
$R_{M,S}(S_1) \subseteq R_{M,S}(S_2)$.
Hence,
$T_{M,A,S}(S_1) = S \cup S_1 \cup R_{A,S}(S_1) \cup R_{M,S}(S_1) \subseteq S \cup S_2 \cup R_{A,S}(S_2) \cup R_{M,S}(S_2) = T_{M,A,S}(S_2)$.
Therefore, $T_{M,A,S}$ is monotonic. □

Lemma 7 *For any set of states S, control module M and action theory A, if S' is a closure of S w.r.t. M and A then S' is a fixpoint of the function $T_{M,A,S}$ defined in Lemma 6.*

Proof: From the definition of $T_{M,A,S}$ we have that
$T_{M,A,S}(S') = S \cup S' \cup R_{A,S}(S') \cup R_{M,S}(S')$.
Hence, $S' \subseteq T_{M,A,S}(S')$.
On the other hand, by Definition 3.1,
$S \subseteq S'$,
$R_{A,S}(S') \subseteq S'$ and
$R_{M,S}(S') \subseteq S'$.
It implies that $T_{M,A,S}(S') = S \cup S' \cup R_{A,S}(S') \cup R_{M,S}(S') \subseteq S'$.
Hence, $T_{M,A,S}(S') = S'$.
In other words, S' is a fixpoint of $T_{M,A,S}$. □

We now prove the proposition which states that the closure of S w.r.t. M and A is unique.

Proposition 3.1 *For any set of states S, control module M and action theory A, there exists a unique set of states which is the closure of S w.r.t. M and A.*

Proof: Let $T_{M,A,S}$ be the function defined in Lemma 6. Since $T_{M,A,S}$ is monotonic (Lemma 6), $T_{M,A,S}$ has a unique least fixpoint. Let us denote the least fixpoint of $T_{M,A,S}$ by $lfp(T_{M,A,S})$. From the definition of $T_{M,A,S}$ it is clear that

- $S \subseteq lfp(T_{M,A,S})$, and
- If $s \in lfp(T_{M,A,S})$ and a is an action in A that can occur independent of the robot then $Res(a,s) \in R_{A,S}(lfp(T_{M,A,S})) \subseteq lfp(T_{M,A,S})$, and
- if $s \in lfp(T_{M,A,S})$ and there exist a rule in M whose LHS is satisfied by s and $[RHS]s$ is defined then $[RHS]s \in R_{M,S}(lfp(T_{M,A})) \subseteq lfp(T_{M,A,S})$.

Thus, $lfp(T_{M,A,S})$ satisfies the three conditions (i)-(iii) in the Definition 3.1.

From the fact that every closure of S w.r.t. M and A is a fixpoint of $T_{M,A,S}$ (Lemma 7) we can conclude that $lfp(T_{M,A,S})$ is a minimal set of states that satisfies the three conditions (i)-(iii) in the Definition 3.1. In other words, $lfp(T_{M,A,S})$ is the unique closure of S w.r.t. M and A. □

Appendix B – Closure and Correctness of module Go_clockwise_1room

In Example 3.1 we showed in detail the computation of $Closure(S_1, M, A_1)$. We now show the detailed computation of $Closure(S_1, M, A_2)$, $Closure(S_1, M, A_3)$, and $Closure(S_1, M, A_4)$.

- **Computing $Closure(S_1, M, A_2)$:**

(i) It is easy to see that $R_{A_2,S_1}(\emptyset) = \emptyset$ and $R_{M,S_1}(\emptyset) = \emptyset$.
Hence, $T^0_{M,A_2,S_1}(\emptyset) = S_1$.

(ii) Since $hand\_turn\_180$ can be executed in every state $s_{1,j,X}$ in S_1 and $[hand\_turn\_180]s_{1,j,X} = s_{2,j,X}$, we have

$$R_{A_2,S_1}(S_1) = \{s_{2,1,app\_corner}, s_{2,2,app\_corner}, s_{2,3,app\_corner}, s_{2,4,app\_corner}, s_{2,5,\neg app\_corner}, s_{2,1,\neg app\_corner}, s_{2,2,\neg app\_corner}, s_{2,3,\neg app\_corner}, s_{2,4,\neg app\_corner}\}.$$

Let us denote $R_{A_2,S_1}(S_1)$ by Y. Then,

$$\begin{aligned} T^1_{M,A_2,S_1}(\emptyset) = T_{M,A_2,S_1}(S_1) &= S_1 \cup Y \cup \{s_{3,1,app\_corner}, s_{3,2,app\_corner}, \\ &\quad s_{3,3,app\_corner}, s_{3,4,app\_corner}, s_{3,5,\neg app\_corner}\} \\ &= X_1 \cup Y \end{aligned}$$

(iii) The rule **if** $just\_started, wall\_on\_rt$ **then** $delete\_started$ is applicable in $s \in X_1 \setminus S_1$ and the rule **if** $white\_door\_on\_rt$ **then** $turn\_180$ is applicable in $s \in Y$. However, the resulting set of states after executing **if** $white\_door\_on\_rt$ **then** $turn\_180$ are exactly S_1. Hence,

$$\begin{aligned} T^2_{M,A_2,S_1}(\emptyset) = T_{M,A_2,S_1}(X_1 \cup Y) &= X_1 \cup Y \cup \{s'_{3,1,app\_corner}, s'_{3,2,app\_corner}, \\ &\quad s'_{3,3,app\_corner}, s'_{3,4,app\_corner}, s'_{3,5,\neg app\_corner}\} \\ &= X_2 \cup Y \end{aligned}$$

(iv) The rule **if** $\neg just\_started, wall\_on\_rt$ **then** $go\_forward$ is applicable in the newly added states. Hence,

$$\begin{aligned} T^3_{M,A_2,S_1}(\emptyset) = T_{M,A_2,S_1}(X_2 \cup Y) &= X_2 \cup Y \cup \{s'_{4,1,\neg app\_corner}, s'_{4,2,\neg app\_corner}, \\ &\quad s'_{4,3,\neg app\_corner}, s'_{4,4,\neg app\_corner}, s'_{1,2,\neg app\_corner}, \\ &\quad s'_{1,4,\neg app\_corner}, s'_{1,5,\neg app\_corner}\} \\ &= X_3 \cup Y \end{aligned}$$

(v) The rule **if** $corridor\_on\_rt$ **then** $turn\_rt\_90$ can be applied in the states $s'_{4,j,\neg app\_corner}$ and the rule **if** $\neg just\_started, black\_door\_on\_rt$ **then** $HALT$ is applicable in the states $s'_{1,j,\neg app\_corner}$. Also, the exogenous action $hand\_turn\_180$ can be executed in the states $s'_{1,j,\neg app\_corner}$. Hence,

$$\begin{aligned} T^4_{M,A_2,S_1}(\emptyset) = T_{M,A_2,S_1}(Y_3) &= X_3 \cup Y \cup \{s'_{3,1,\neg app\_corner}, s'_{3,2,\neg app\_corner}, \\ &\quad s'_{3,3,\neg app\_corner}, s'_{3,4,\neg app\_corner}, s'_{2,2,\neg app\_corner}, \\ &\quad s'_{2,4,\neg app\_corner}, s'_{2,5,\neg app\_corner}\} \\ &= X_4 \cup Y_1 \end{aligned}$$

where $Y_1 = Y \cup \{s'_{2,2,\neg app\_corner},\ s'_{2,4,\neg app\_corner},\ s'_{2,5,\neg app\_corner}\}$

(vi) The rules **if** $\neg just\_started, wall\_on\_rt$ **then** $go\_forward$ and **if** $white\_door\_on\_rt$ **then** $turn\_180$ are applicable in $(X_4 \cup Y_1) \setminus (X_3 \cup Y)$. Hence, we have

$$T^5_{M,A_1,S_1}(\emptyset) = T_{M,A_1,S_1}(X_4 \cup Y_1) = Y_4 \cup \{s'_{1,3\neg app\_corner},\ s'_{1,1\neg app\_corner}\} = X_5 \cup Y_1$$

(vii) The exogenous actions $hand\_turn\_180$ is applicable in the states in $X_5 \setminus X_4$. Hence,

$$T^6_{M,A_1,S_1}(\emptyset) = T_{M,A_1,S_1}(X_5 \cup Y_1) = Y_5 \cup \{s'_{2,3\neg app\_corner},\ s'_{2,1\neg app\_corner}\} = X_5 \cup Y_2$$

where $Y_2 = Y_1 \cup \{s'_{2,3\neg app\_corner},\ s'_{2,1\neg app\_corner}\}$

(viii) It is easy to check that $T_{M,A_1,S_1}(X_5 \cup Y_2) = X_5 \cup Y_2$. Hence

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

– **Computing** $Closure(S_1, M, A_3)$**:**

As we can see from the computation of $Closure(S_1, M, A_2)$ that the difference between $Closure(S_1, M, A_2)$ and $Closure(S_1, M, A_1)$ depends only on the exogenous actions. Therefore, in the following, instead of computing $Closure(S_1, M, A_3)$, we will only compute the difference between $Closure(S_1, M, A_2)$ and $Closure(S_1, M, A_3)$.
It is easy to see in the first two steps, i.e., for $i = 0, 1$, $T^i_{M,A_3,S_1}(\emptyset) = T^i_{M,A_2,S_1}(\emptyset)$. Hence, $T^1_{M,A_3,S_1}(\emptyset) = X_1 \cup Y$.
Since $R_{A_3,S_1}(X_1 \cup Y) = \{s_{4,1,\neg app\_corner},\ s_{4,2,\neg app\_corner}, s_{4,3,\neg app\_corner},\ s_{4,4,\neg app\_corner}\}$, we have
$T^2_{M,A_3,S_1}(\emptyset) = X_2 \cup Y \cup \{s_{4,1,\neg app\_corner},\ s_{4,2,\neg app\_corner}, s_{4,3,\neg app\_corner},\ s_{4,4,\neg app\_corner}\}$.
Here the rule **if** $corridor\_on\_rt$ **then** $turn\_rt\_90$ takes the robot from the state $s_{4,j,\neg app\_corner}$ to $s_{3,j,\neg app\_corner}$. Hence, we have
$T^3_{M,A_3,S_1}(\emptyset) = X_3 \cup Y \cup Z$ where
$Z = \{s_{4,1,\neg app\_corner},\ s_{4,2,\neg app\_corner}, s_{4,3,\neg app\_corner},\ s_{4,4,\neg app\_corner}\} \cup$
$\{s_{3,1,\neg app\_corner},\ s_{3,2,\neg app\_corner}, s_{3,3,\neg app\_corner},\ s_{3,4,\neg app\_corner}\}$
Finally, we get

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

– **Computing** $Closure(S_1, M, A_4)$**:**

Since every state which is reached by the exogenous action *coner_change* is contained in $Closure(S_1, M, A_3)$ we have $Closure(S_1, M, A_4) = Closure(S_1, M, A_3)$.

Correctness of module Go_clockwise_1room

For each possible states of the action theory about the control module $M = Go\_clockwise\_1room$, the following table lists $\mathcal{U}_M(s)$ and the state that the control module terminates at. (It should be noted that to reduce the number of states the fluent $at(other)$ is used as an abstraction. Because of that some of the states in our language are actually a collection of states, resulting in the fact that for a state s, where $at(other)$ is true, $\mathcal{U}_M(s)$ may not be unique. This is not because of non-deterministic actions – we don't allow them, but because of our abstraction.) It is easy to see that the goal is satisfied in those terminating states.

Initial state (s)	Unfolding control module M with respect to the state s ($\mathcal{U}_M(s)$)	State reached after executing $\mathcal{U}_M(s)$
s_{1,j,app_corner} $(j = 1, 3)$	$go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,5,\neg app\_corner}$
$s_{1,2,app\_corner}$	$go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,3,\neg app\_corner}$
$s_{1,4,app\_corner}$	$go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,1,\neg app\_corner}$
$s_{1,1,\neg app\_corner}$ $(j = 1, 3)$	$go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,5,\neg app\_corner}$
$s_{1,2,\neg app\_corner}$	$go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,3,\neg app\_corner}$
$s_{1,4,\neg app\_corner}$	$go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,1,\neg app\_corner}$
$s_{1,5,\neg app\_corner}$	$go\_forward; delete\_started; go\_forward;$	$s'_{1,k,\neg app\_corner}$ $(k \in \{2, 4, 5\})$
$s'_{1,j,\neg app\_corner}$ $(j = 1, \ldots, 5)$	[]	$s'_{1,j,\neg app\_corner}$
s'_{1,j,app_corner} $(j = 1, \ldots, 4)$	[]	s'_{1,j,app_corner}

Initial State (s)	Unfolding control module M with respect to the state s ($\mathcal{U}_M(s)$)	State Reached after Executing $\mathcal{U}_M(s)$
s_{2,j,app_corner} $(j = 1, 3)$	$turn\_180; go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,5,\neg app\_corner}$
$s_{2,2,app\_corner}$	$turn\_180; go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,3,\neg app\_corner}$
$s_{2,4,app\_corner}$	$turn\_180; go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,1,\neg app\_corner}$
$s_{2,1,\neg app\_corner}$ $(j = 1, 3)$	$turn\_180; go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,5,\neg app\_corner}$
$s_{2,2,\neg app\_corner}$	$turn\_180; go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,3,\neg app\_corner}$
$s_{2,4,\neg app\_corner}$	$turn\_180; go\_forward; delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,1,\neg app\_corner}$
$s_{2,5,\neg app\_corner}$	$turn\_180; go\_forward; delete\_started; go\_forward;$	$s'_{1,k,\neg app\_corner}$ $(k \in \{2, 4, 5\})$
$s'_{2,j,\neg app\_corner}$ $(j = 1, \ldots, 5)$	$turn\_180$	$s'_{1,j,\neg app\_corner}$
s'_{2,j,app_corner} $(j = 1, \ldots, 4)$	$turn\_180$	s'_{1,j,app_corner}

Initial State (s)	Unfolding control module M with respect to the state s ($\mathcal{U}_M(s)$)	State Reached after Executing $\mathcal{U}_M(s)$
s_{3,j,app_corner} $(j = 1, 3)$	$delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,5,\neg app\_corner}$
$s_{3,2,app\_corner}$	$delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,3,\neg app\_corner}$
$s_{3,4,app\_corner}$	$delete\_started; go\_forward; turn\_right\_90; go\_forward$	$s'_{1,1,\neg app\_corner}$
$s_{3,j,\neg app\_corner}$ $(j = 1, 3)$	$delete\_started; go\_forward$	$s'_{1,5,\neg app\_corner}$
$s_{3,2,\neg app\_corner}$	$delete\_started; go\_forward$	$s'_{1,3,\neg app\_corner}$
$s_{3,4,\neg app\_corner}$	$delete\_started; go\_forward$	$s'_{1,1,\neg app\_corner}$
$s_{3,5,\neg app\_corner}$	$delete\_started; go\_forward$	$s'_{1,k,\neg app\_corner}$ $(k \in \{2, 4, 5\})$
s'_{3,j,app_corner} $(j = 1, 3)$	$go\_forward; turn\_right\_90; go\_forward$	$s'_{1,5,\neg app\_corner}$
$s'_{3,2,app\_corner}$	$go\_forward; turn\_right\_90; go\_forward$	$s'_{1,3,\neg app\_corner}$
$s'_{3,4,app\_corner}$	$go\_forward; turn\_right\_90; go\_forward$	$s'_{1,1,\neg app\_corner}$
$s'_{3,j,\neg app\_corner}$ $(j = 1, 3)$	$go\_forward$	$s'_{1,5,\neg app\_corner}$
$s'_{3,2,\neg app\_corner}$	$go\_forward$	$s'_{1,3,\neg app\_corner}$
$s'_{3,4,\neg app\_corner}$	$go\_forward$	$s'_{1,1,\neg app\_corner}$
$s'_{3,5,\neg app\_corner}$	$go\_forward$	$s'_{1,k,\neg app\_corner}$ $(k \in \{2, 4, 5\})$

Initial State (s)	Unfolding control module M with respect to the state s ($\mathcal{U}_M(s)$)	State Reached after Executing $\mathcal{U}_M(s)$
$s_{4,1,\neg app\_corner}$ ($j = 1,3$)	$turn\_rt\_90; delete\_started; go\_forward;$	$s'_{1,5,\neg app\_corner}$
$s_{4,2,\neg app\_corner}$	$turn\_rt\_90; delete\_started; go\_forward;$	$s'_{1,3,\neg app\_corner}$
$s_{4,4,\neg app\_corner}$	$turn\_rt\_90; delete\_started; go\_forward;$	$s'_{1,1,\neg app\_corner}$
$s'_{4,1,\neg app\_corner}$ ($j = 1,3$)	$turn\_rt\_90; go\_forward;$	$s'_{1,5,\neg app\_corner}$
$s'_{4,2,\neg app\_corner}$	$turn\_rt\_90; go\_forward;$	$s'_{1,3,\neg app\_corner}$
$s'_{4,4,\neg app\_corner}$	$turn\_rt\_90; go\_forward;$	$s'_{1,1,\neg app\_corner}$

Appendix C – Proofs in Section 4.1

In this section we prove Lemma 1 and 2 and Theorem 1 from Section 4.1. First we start with some notations.

We use a cost function whose domain is a 3-tuple of two states (the initial state and the final state) and a plan and whose range is a positive integer. Intuitively, by $cost(s, s', P) = n$, we mean that n is the cost of executing P in state s to reach state s'. When P is a single action, we require the cost function to satisfy the following conditions:

- $0 < cost(s, s', a) < \infty$ if $s' = Res(a, s)$, and
- $cost(s, s', a)$ is undefined if $s' \neq Res(a, s)$,

Given a cost function $cost$ of A, the $cost$ function from a state s to a state s' by means of a plan $Q = a \circ P$ is defined inductively as follows:

- $cost(s, s, []) = 0$, and

- $cost(s, s', Q) = cost(s, [a]s, a) + cost([a]s, s', P)$, and
- $cost(s, s', Q)$ is undefined if $s' \neq [Q]s$.

A plan P is called a minimal cost plan from s to s' if

$$cost(s, s', P) = min\{cost(s, s', P) \ : \ [P]s = s'\}.$$

If P is a plan which achieves G from s then we define $cost(s, G, P) = cost(s, [P]s, P)$. A plan P is called a minimal cost plan which achieves G from a state s if

$$cost(s, G, P) = min\{cost(s, G, P') \ : \ P' \text{ achieves } G \text{ from } s\}.$$

Lemma 1 *Let M be a sound and complete control module w.r.t. (G, S). Then, for every $s \in S$ if the rule* **if** LHS **then** RHS *is applicable in s and P is a minimal cost plan that achieves G from s such that $P = RHS \circ Q$ then Q is a minimal cost plan that achieves G from $[RHS]s$.*

Proof: Let $s' = [RHS]s$. Assume the contrary, i.e., Q is not a minimal cost plan from s' to G. That means, there exists a plan L which achieves G from s' with $cost(s', G, L) < cost(s', G, Q)$.
Since L is a plan achieving G from s', $P' = RHS \circ L$ is a plan achieving G from s.
Furthermore, $cost(s, G, P') = cost(s, s', RHS) + cost(s', G, L) < cost(s, s', RHS) + cost(s'G, Q) = cost(s, G, P)$.
This implies that P is not a minimal cost plan achieving G from s. This contradicts our assumption. Hence, Q is a minimal cost plan that achieves G from $[RHS]s$. □

Lemma 2 *If M is sequential w.r.t. a set of states S and is sound and complete w.r.t. a goal G and S then $\mathcal{U}_M(s)$ is finite for $s \in S$.*

Proof: Let

$S_1 = \{s \in S \ : \ \text{all rules applicable in } s \text{ are termination control rules } \}$,

$S_2 = \{s \in S \ : \ \text{there exists a simple control rule in } M \text{ which is applicable in } s\}$, and

$S_3 = S \setminus (S_1 \cup S_2)$.

It is easy to see that S_1, S_2, S_3 are pairwise disjoint.

Since M is complete w.r.t (G, S), $S_3 = \emptyset$.

We will prove the lemma by contradiction.
Let us assume that $\mathcal{U}_M(s)$ is infinite for some $s \in S$.

By Definition 3.3, $\mathcal{U}_M(s) = []$ for $s \in S_1$. Hence, $s \in S_2$.

Let α be the compound action that represents the concurrent execution of the RHS of all rules which are applicable in s and $s_1 = [\alpha]s$. (Note that, since, M is sequential, α is simply a sequence of actions.)

Since $\mathcal{U}_M(s) = \alpha \circ \mathcal{U}_M([\alpha]s)$, $\mathcal{U}_M(s_1)$ is infinite.

Thus, $s_1 \in S_2$.

Similarly, we can prove that there exists an infinite sequence of states $s_0 = s, s_1, \ldots$, such that for $0 \leq i$

(i) $s_i \in S_2$, and
(ii) $\mathcal{U}_M(s_i)$ is infinite, and
(iii) $s_{i+1} = [\alpha_i]s_i$ where α_i is the RHS of all rules applicable in s_i.
Since M is sequential, the RHS of all rules applicable in s_i is the same. Thus, if the rule **if** LHS_i **then** RHS_i is applicable in s_i then $\alpha_i = RHS_i$.

Since S_2 is finite there exists a number $1 \leq k$ such that $s_i \notin \{s_0, \ldots, s_{i-1}\}$ for $i < k$ and $s_k \in \{s_0, \ldots, s_{k-1}\}$.

Without loss of generality we can assume that $s_k = s_t$ for some $0 \leq t < k$. Because M is sound there are minimal cost plans $P_i = RHS_i \circ Q_i$ from s_i to G for $(t \leq i \leq k)$.
Since Q_i is a minimal cost plan from s_{i+1} to G (Lemma 1) we have

$$cost(s_{t+i+1}, G, Q_{t+i}) = cost(s_{t+i+1}, s_{t+i+2}, RHS_{t+i+1}) + cost(s_{t+i+2}, G, Q_{t+i+1})$$

for $(0 \leq i < k)$ and

$$cost(s_t, G, Q_k) = cost(s_t, s_{t+1}, RHS_t) + cost(s_{t+1}, G, Q_t)$$

So, we have

$$\sum_{i=t}^{k} cost(s_i, G, Q_i) = \sum_{i=t}^{k} cost(s_i, s_{i+1}, RHS_i) + \sum_{i=t}^{k} cost(s_i, G, Q_i)$$

where $s_{k+1} = s_t$. Hence,

$$\sum_{i=t}^{k} cost(s_i, s_{i+1}, RHS_i) = 0$$

which contradicts the fact that $cost(s_i, s_{i+1}, RHS_i) > 0$ for every i because $[RHS_i]s_i = s_{i+1}$. So, our assumption is incorrect. Thus, $\mathcal{U}_M(s)$ is finite for every $s \in S$.

□

Theorem 1 *Let A be an action theory. Consider a pair (M, S), where S is a set of complete states and 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 and complete w.r.t. G and S then M achieves G from S w.r.t. A.*

Proof: Let x be a state in S. Since M is sequential, $\mathcal{U}_M(x) = RHS_1 \circ RHS_2 \circ \ldots$ where RHS_{i+1} is the right hand side of the rule which is applicable in $[RHS_1 \circ \ldots \circ RHS_i]x$. From Lemma 2 we have that $\mathcal{U}_M(x)$ is finite. Furthermore, a_F^M is not contained in any RHS of rules in M. Hence, the last action in $\mathcal{U}_M(x)$ is not a_F^M. We now prove inductively over $|\mathcal{U}_M(x)|$, the number of actions in $\mathcal{U}_M(x)$, that the state $x' = [\mathcal{U}_M(x)]x$ satisfies G.

- *Base case:* $|\mathcal{U}_M(x)| = 0$.
  It implies that the right hand side of all rules applicable in x is HALT.
  Let r be a termination control rule **if** LHS **then** $HALT$ in M which is applicable in x.
  Because M is sound w.r.t (G, S), G is satisfied by LHS.
  Since r is applicable in x, LHS is satisfied in x.
  This means, G is satisfied in x too.
  Because $x' = []x = [\mathcal{U}_M(x)]x$, the base case is proved. (i)

- *Inductive case:* Assuming that $[\mathcal{U}_M(x)]x$ satisfies G for all $x \in S$, when $|\mathcal{U}_M(x)| \leq n$; we now prove that $[\mathcal{U}_M(z)]z$ satisfies G for all $z \in S$, when $|\mathcal{U}_M(z)| = n + 1$.
  Since M is sequential, we can assume that the right hand side of all rules which are applicable in z is RHS.
  Since $|\mathcal{U}_M(z)| > 0$, RHS is not HALT.
  Hence, $1 \leq |RHS|$.
  Consider the state $y = [RHS]z$.
  It follows from $\mathcal{U}_M(z) = RHS \circ \mathcal{U}_M(y)$ and $1 \leq |RHS|$ that $|\mathcal{U}_M(y)| \leq n$.
  Hence, by inductive hypothesis G is satisfied in $[\mathcal{U}_M(y)]y$.
  Since $[\mathcal{U}_M(z)]z = [RHS \circ \mathcal{U}_M(z)]z = [\mathcal{U}_M(y)]y$, G is satisfied in $[\mathcal{U}_M(z)]z$.
  The inductive case is proved. (ii)

From (i)-(ii), we can conclude that $[\mathcal{U}_M(x)]x$ satisfies G for all $x \in S$. Hence, M achieves the goal G. □

Correctness of Algorithm 4.2

In the following, we will prove the correctness of Algorithm 4.2 w.r.t. an action theory A and a reachable goal G. First, let us present some notations which will be used in the proofs. Given an action theory A, and a set of states S, let Σ denote $Closure(S, A, A)$. A goal G is said to be reachable from A and S, if for every state s in Σ, there exists a plan achieving G from s.

It is easy to see that the Algorithm 4.2 will halt if the step 2 is not repeated infinitely many times. If S_{in}, S_{out}, or M change in step 2, then the algorithm will go to the next round of step 2. We distinguish these sets in every round by the following notations:

$S_{in}^i = S_{in}$ after the i-th round of step 2,

$S_{out}^i = S_{out}$ after the i-th round of step 2, and

$M^i = M$ after the i-th round of step 2.

where $S_{in}^0 = S \cup \{s' : s'$ is reachable from some state s in S by applying sequences of exogenous actions$\}$, and $S_{out}^0 = \emptyset$.

Note that in step 2, we pick up a state $s \in S_{in} \setminus S_{out}$ and then add it to S_{out} (either in step 2.2 or step 2.6). Furthermore, we never delete a state from S_{in} and we add to S_{in} all the states which are in the closure of S_{in} w.r.t. M_i and A in step 2.6. Hence, we have the following lemma.

Lemma 8 *(i) For every integer t, $S_{out}^t \subseteq S_{in}^t \subseteq \Sigma$.*
(ii) For every integer t and l such that $t < l$, $S_{in}^t \subseteq S_{in}^l$.
(iii) For every integer t, if $S_{out}^t \neq S_{in}^t$ then $S_{out}^t \subset S_{out}^{t+1}$.
(iv) For every integer t, $S_{in}^t = Closure(S, M^t, A)$.

Proof: The first three items of the lemma are obvious based on the above discussion. We will next prove the fourth item by induction over t. (The basic idea behind the proof is that when we add a new rule to M in the t-th iteration of step 2.6 only one state in S_{in}^t satisfies the LHS of this rule.)

(i) **Base case** $t = 0$**:**
Since M^0 contains only one rule **if** G **then** $HALT$, $Closure(S, M^0, A) = S \cup \{s' : s'$ is reachable from some state s in S by applying sequences of exogenous actions$\}= S_{in}^0$. Therefore, $S_{in}^0 = Closure(S, M^0, A)$. The base case is proved. (*)

(ii) **Inductive step:**
Assuming that we have proved that $S_{in}^t = Closure(S, M^t, A)$ for $t \leq n$.
We will now prove that $S_{in}^{n+1} = Closure(S, M^{n+1}, A)$.
Assuming that we have selected a state $s \in S_{in}^n$ in step 2.1.
If s satisfies G then $S_{in}^{n+1} = S_{in}^n$ and $M_n = M_{n+1}$.
By inductive hypothesis, $S_{in}^n = Closure(S, M^n, A)$ and therefore $S_{in}^{n+1} = Closure(S, M^{n+1}, A)$.
Otherwise, let us assume that the plan $a_1, \ldots, a_m$ is a minimal plan achieving G from s in step 2.3 and i is the number satisfying the condition in step 2.4.
First, it is easy to see that $T_{M^n,S,A}(X) \subseteq T_{M^{n+1},S,A}(X)$ because $M^n \subseteq M^{n+1}$. Hence, $Closure(S, M^n, A) \subseteq Closure(S, M^{n+1}, A)$.

Thus, $S^n_{in} \subseteq Closure(S, M^{n+1}, A)$. Furthermore, for a state $s' \in S^{n+1}_{in} \setminus S^n_{in}$ there are two cases:

(a) $s' = [a_1, \ldots, a_j]s$ for some j, $0 \leq j < i$.
Since **if** $[a_1, \ldots, a_{j-1}]s$ **then** a_j belongs to M^{n+1} and $s \in S^n_{in}$, $s' \in Closure(S, M^{n+1}, A)$.

(b) s' is reachable from $[a_1, \ldots, a_j]s$ for some $j < i$ by applying sequences of exogenous actions. Thus, $s' \in Closure(S, M^{n+1}, A)$.

In both cases we have that if $s' \in S^{n+1}_{in} \setminus S^n_{in}$ then $s' \in Closure(S, M^{n+1}, A)$, i.e., $S^{n+1}_{in} \setminus S^n_{in} \subseteq Closure(S, M^{n+1}, A)$.
Hence, $S^{n+1}_{in} = S^n_{in} \cup (S^{n+1}_{in} \setminus S^n_{in}) \subseteq Closure(S, M^n, A) \cup Closure(S, M^{n+1}, A) \subseteq Closure(S, M^{n+1}, A)$ (1)
We now prove that $Closure(S, M^{n+1}, A) \subseteq S^{n+1}_{in}$.
Recall that, $T_{M^{n+1},S,A}(S^{n+1}_{in}) = S \cup S^{n+1}_{in} \cup R_{A,S}(S^{n+1}_{in}) \cup R_{M^{n+1},S}(S^{n+1}_{in})$.
Since $R_{A,S}(S^{n+1}_{in}) = R_{A,S}(S^n_{in}) \cup R_{A,S}(S^{n+1}_{in} \setminus S^n_{in})$, and because of step 2.6 $R_{A,S}(S^{n+1}_{in}) \subseteq S^{n+1}_{in}$.
Furthermore, in step 2.6, if a rule r : **if** x **then** a is added to M^{n+1} then x is a complete state and therefore, there exists no other state x' in S^{n+1}_{in} such that r is applicable in x'.
Hence, $R_{M^{n+1},S}(S^{n+1}_{in}) \subseteq S^{n+1}_{in}$
So, $T_{M^{n+1},S,A}(S^{n+1}_{in}) \subseteq S^{n+1}_{in}$.
It follows from $S^{n+1}_{in} \subseteq T_{M^{n+1},S,A}(S^{n+1}_{in})$ and $T_{M^{n+1},S,A}(S^{n+1}_{in}) \subseteq S^{n+1}_{in}$ that S^{n+1}_{in} is a fixpoint of $T_{M^{n+1},S,A}$.
Thus, $Closure(S, M^{n+1}, A) \subseteq S^{n+1}_{in}$. (2)
From (1) and (2), we can conclude that $Closure(S, M^{n+1}, A) = S^{n+1}_{in}$. The inductive step is proved. (**)

The lemma is proved from (*)-(**). □

Lemma 9 *Given S, A, and G. If G can be achieved from any state in Σ then the algorithm 4.2 halts.* □

Proof: Because A is finite, Σ is finite, and G can be achieved from every state in Σ, it is clear that there exists some n such that $S^n_{out} = S^n_{in}$. Hence, the algorithm will stop after finitely many rounds of step 2.

Lemma 10 *Given S, A, and G. If G can be achieved from every state in Σ then for every integer n and for every $s \in S^n_{out}$,*

(i) there exists one and only one rule r in M^n which is applicable in s, and
(ii) $\mathcal{U}_{M^n}(s)$ is finite, and
(iii) G is satisfied in $[\mathcal{U}_{M^n}(s)]s$.

Proof: The lemma is trivial for $n = 0$ because $S^0_{out} = \emptyset$. We will now prove it for $n \geq 1$ by induction over n.

(i) *Base case: n=1.* Consider an arbitrary $s \in S^1_{out}$. There are two cases:

- **s is added to S^1_{out} in the step 2.2.**
  It means that s satisfies G.
  Hence, the rule **if** G **then** $HALT$ is applicable in s (which is the only rule in M^1).
  Thus, $\mathcal{U}_{M^1}(s) = []$ (and therefore $\mathcal{U}_{M^1}(s)$ is finite),
  and G is satisfied in $[\mathcal{U}_{M^1}(s)]s$. (1)
- **s is added to S^1_{out} in the step 2.6.**
  It implies that we have picked some state $s' \in S^0_{in}$ in step 2.1,
  found a minimal plan $P = a_1; \ldots; a_k$ which achieves G from s' in step 2.3 and then added $s = [a_1; \ldots; a_j]s'$ to S^1_{out} in the step 2.6 for some j where $0 \leq j < i$, $i = k+1$ as determined in step 2.4.
  Let $s_j = [a_1, \ldots, a_j]s'$.
  Because P is a minimal plan, $s_{j_1} \neq s_{j_2}$ for $j_1 \neq j_2$.
  Furthermore, because of step 2.5, for each s_j, $j < k$ there exists a rule **if** s_j **then** a_{j+1} in M^1 which is applicable in s_j and
  since s_k satisfies G, **if** G **then** $HALT$ is applicable in s_k.
  It is easy to see that $\mathcal{U}_{M^1}(s_j) = a_{j+1} \circ \mathcal{U}_{M^1}(s_{j+1})$ for $j < k$ and $\mathcal{U}_{M^1}(s_k) = []$.
  Hence, $\mathcal{U}_{M^1}(s_j)$ is finite for $1 \leq j \leq k$.
  Furthermore, because $[\mathcal{U}_{M^1}(s_j)]s_j = [a_1, \ldots, a_k]s'$, G is satisfied in $[\mathcal{U}_{M^1}(s_j)]s_j$ for $1 \leq j \leq k$. Thus, the lemma holds for every $s \in S^1_{out}$. (2)

  The base case is proved because of (1) and (2). (*)

(ii) *Inductive step:* Assuming that we have proved the lemma for $n \leq t$. We now prove that for $n = t+1$.
From the steps 2.5 and 2.6, it is clear that if a rule r is added to M^{t+1}, i.e., $r \in M^{t+1} \setminus M^t$, then there is a state $s \in S^{t+1}_{out}$ and r has the form **if** s **then** a for some action a and $s \notin S^t_{out}$.
Thus, for every s in S^t_{out}, the algorithm does not add new rule to M^{t+1} which is applicable in s.
In other words, for $s \in S^t_{out}$, the lemma holds because of the inductive hypothesis. (3)
We now prove the lemma for $s \in S^{t+1}_{out} \setminus S^t_{out}$. Again, since s is added to S^{t+1}_{out} only in steps 2.2 or 2.6 there are two cases.

- **s is added to S^{t+1}_{out} in step 2.2.**
  Similar to the first case of the base case, we can show that $\mathcal{U}_{M^{t+1}}(s)$ is finite, and G is satisfied in $[\mathcal{U}_{M^{t+1}}(s)]s$. (4)
- **s is added to S^{t+1}_{out} in step 2.6.**
  It implies that we have selected a state s' in S^t_{in} and
  found a minimal plan $P = a_1; \ldots; a_k$ which achieves G from s
  such that $s = [a_1; \ldots; a_j]s'$ for some $j \leq i$ where $i = k+1$ if $[a_1; \ldots; a_k]s' \notin S^t_{out}$ and $i < k$ if $[a_1; \ldots; a_k]s' \in S^s_{out}$.
  If $i = k+1$ then similar to the second case of the base case we can prove

that $\mathcal{U}_{M^{t+1}}(s_j) = a_{j+1} \circ \mathcal{U}_{M^{t+1}}(s_{j+1})$ for $j < k$ and $\mathcal{U}_{M^{t+1}}(s_k) = []$. (5)
Otherwise, let i be the number in the step 2.4 and $s_j = [a_1, \ldots, a_j]s'$ for $j < i$.
Since P is a minimal plan, $s_{j_1} \neq s_{j_2}$ for $j_1 \neq j_2$.
It is easy to see that $\mathcal{U}_{M^{t+1}}(s_j) = a_{j+1} \circ \mathcal{U}_{M^{t+1}}(s_{j+1})$ for $j < i$ and $\mathcal{U}_{M^{t+1}}(s_i) = \mathcal{U}_{M^t}(s_i)$.
By inductive hypothesis, $\mathcal{U}_{M^t}(s_i)$ is finite and G is satisfies in $[\mathcal{U}_{M^t}(s_i)]s_i$.
Hence, $\mathcal{U}_{M^{t+1}}(s_j)$ is finite for $0 \leq j \leq i$ and G is satisfied in $[\mathcal{U}_{M^{t+1}}(s_j)]s_j$ for $0 \leq j \leq i$. (6)
From (5) and (6) we can conclude that for every $s \in S^{t+1}_{out} \setminus S^t_{out}$, there exists one and only one rule r in M^{t+1} which is applicable in s, $\mathcal{U}_{M^{t+1}}(s)$ is finite, and G is satisfied in $[\mathcal{U}_{M^{t+1}}(s)]s$. (7)
The inductive step is proved by (3)-(7). (**)

The lemma is proved by (*)-(**). $\Box$

Proposition 4.3 *Given a goal G, a set of states S and an action theory A if G is achievable from all states in $Closure(S, A, A)$, then Algorithm 4.2 generates a control module M such that M achieves G from $Closure(S, M, A)$.*

Proof: It follows directly from Lemma 9 that the algorithm will halt after finitely many rounds of step 2, say n. Furthermore, Lemma 8 implies that $Closure(S, M, A) = S^n_{in} = S^n_{out}$. So, for each state $s \in Closure(S, M, A)$, there exists i, $1 \leq i \leq n$, such that $s \in S^i_{out}$ and $s \notin S^{i-1}_{out}$. Hence, by Lemma 10 there exists a rule r in M which is applicable in s and therefore $\mathcal{U}_M(s)$ does not end with a^F_M. Also, by Lemma 10, $\mathcal{U}_M(s)$ is finite and G is satisfied in $\mathcal{U}_M(s)$. Thus, by definition, M achieves G. $\Box$

Appendix D: Proofs of lemmas and theorems in Section 5

Before proving the lemmas and theorems of the section we present an example which shows that a simple soundness condition for the control modules as in Definition 4.2 is not sufficient for control modules with sensing actions.

Example 13.1 Consider a domain description A with the following action

(0.2)	do_c	**causes**	c		
(0.2)	do_c'	**causes**	$\neg c$		
(5)	s_a	**determines**	a	**if**	c
(5)	s_b	**determines**	b	**if**	$\neg c$
(4)	a_1	**causes**	f	**if**	a
(4)	a_2	**causes**	f	**if**	$\neg a$
(3)	b_1	**causes**	f	**if**	b
(5)	b_2	**causes**	f	**if**	$\neg b$

where the number in the parenthesis at the beginning of each line is the cost assigned to the action describing in the same line.

Let M be the following control module.

if $\neg c, u(a), u(b)$ **then** do_c
if $c, u(a), u(b)$ **then** do_c'
if b **then** b_1
if $a, \neg b$ **then** a_1
if $\neg a, \neg b$ **then** a_2
if $u(a), \neg b$ **then** b_2
if $a, u(b)$ **then** a_1
if $\neg a, u(b)$ **then** a_2
if f **then** $HALT$

Let us refer to the set of possible state s which is not empty, i.e., $s \neq \langle \emptyset, \emptyset \rangle$ with S.

It is easy to check that the control module M satisfies the condition that for each $s \in S$ there exists a rule in M which is applicable in s. In other words, M is complete w.r.t S.

We now show that for each rule **if** LHS **then** RHS in M there exists a minimal cost plan achieving f whose first action is RHS. This is easy to check for the last seven rules (1).

Consider the first rule, there are two minimal conditional plans that achieves f from $\langle \emptyset, \{c\} \rangle$. They are:

$C_1 = do\_c;$
s_a

case
$a \quad \rightarrow a_1$
$\neg a \quad \rightarrow a_2$
endcase

and

$C_2 = s\_b$
case
$b \quad \rightarrow b_1$
$\neg b \quad \rightarrow b_2$
endcase

Since $Cost(Unfold(C_1, \{\neg c, a, b\}) = 9.2$ and $Cost(Unfold(C_2, \{\neg c, a, b\}) = 8$ we have that $C_2 \not\leq_{S,\{\neg c\}} C_1$.

Similarly, because $Cost(Unfold(C_1, \{\neg c, a, \neg b\}) = 9.2$ and $Cost(Unfold(C_2, \{\neg c, a, \neg b\}) = 10$ we have that $C_1 \not\leq_{S,\{\neg c\}} C_2$.

Thus, both plans C_1 and C_2 are minimal cost plans achieving f from $\langle \emptyset, \{c\} \rangle$ (2).

This means that there exists a minimal cost plan achieving f with the first action is the RHS of the first rule.

Similarly, we can show that the two plans

$D_1 = do\_c'$;
s_b
case
$b \quad \rightarrow b_1$
$\neg b \quad \rightarrow b_2$
endcase

and

$D_2 = s\_a$
case
$a \quad \rightarrow a_1$
$\neg a \quad \rightarrow a_2$
endcase

are two minimal cost plans achieving f from $\langle \{c\}, \emptyset \rangle$ (3).

Thus, for each control rule r in M there exists a minimal cost plan that achieves f from the state satisfying the LHS of r whose first action is the RHS of r (because (1)-(3)).

However, the control module M containing these two rules will not achieve f from every possible state which is not empty because $\mathcal{U}_M(\langle \{c\}, \emptyset \rangle)$ and $\mathcal{U}_M(\langle \emptyset, \{\neg c\} \rangle)$ are infinite. □

We now prove the lemmas and theorems in Section 5. In the following by C or C_i we denote a conditional plan.

Lemma 3 *Let C be a minimal conditional plan w.r.t (G, s). Then, C is empty or C starts with an action.*

Proof: The lemma is trivial if C is empty. Consider the case C is not empty. Then C can be represented as the sequence $C_1; \ldots; C_k$, where C_i's are either non-empty sequences of actions or are case plans. We will show that C_1 is a sequence of actions.

Assuming the contrary, C_1 is a case plan of the form

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

Since $\{p_{1,1}, \ldots, p_{1,m_1}\}, \ldots, \{p_{n,1}, \ldots, p_{n,m_n}\}$ are mutually exclusive, there exists at most one i such that s satisfies $p_{i,1}, \ldots, p_{i,m_i}$. In that case, the sub-plan $C' = c_i; C_2; \ldots; C_k$ achieves G from s. Otherwise, the sub-plan $C' = C_2; \ldots; C_k$ achieves G from s. In both cases, it contradicts the fact that C is minimal w.r.t (G, s). Hence, C_1 is a sequence of actions. Because C_1 is not empty, the lemma is proved. $\Box$

Lemma 5 *Let S be a set of states and C be a minimal conditional plan that achieves G from s. If C contains a case plan and $pref(P)$ is knowledge-preserving in s then*

(i) $pref(P)$ contains a sensing action and
(ii) s is an incomplete state.

Proof: From Lemma 3, we know that C starts with an action (or a sequence of actions). Hence, it is easy to see that C has the following form

$C = C_0;$
Case
$s_1 \rightarrow C_1$
$\vdots$
$s_n \rightarrow C_n$
Endcase
D

where C_0 is a sequence of actions and D is a conditional plan.

We first prove that $\{s_1, \ldots, s_n\} \subseteq \Phi(C_0, s)$.
Assume the contrary; i.e., $\Phi(C_0, s)$ does not contain s_i for some $s_i \in \{s_1, \ldots, s_n\}$.

Then
$C' = C_0;$
Case
$s_1 \rightarrow C_1$
$\vdots$
$s_{i-1} \rightarrow C_{i-1}$
$s_{i+1} \rightarrow C_{i+1}$
$s_n \rightarrow C_n$
Endcase
D
is a sub-plan of C which achieves G from s. This contradicts our assumption that C is minimal w.r.t (G, s).

We now prove that C_0 contains a sensing action. Assume the contrary, i.e., C_0 does not contain a sensing action. By Lemma 4, $|\Phi(C_0)| = 1$. Without loss of generality, we can assume that $\Phi(C_0) = \{s_1\}$. Again, the sub-plan $C' = C_0; C_1; D$ of C achieves G from s which contradicts the fact that C is minimal w.r.t (G, s). *Hence, C_0 contains a sensing action.*

We now prove that s is an incomplete state. Again, we prove by contradiction. Assume that s is a complete state.

Let $C_0 = a_1, \ldots, a_l, \ldots, a_n$ and a_l be the first sensing action in C_0. It is easy to see that $\Phi(a_1; \ldots; a_{l-1}, s) = \Phi(a_1; \ldots; a_l, s)$. Hence, the plan
$C' = a_1, \ldots, a_{l-1}, a_{l+1}, \ldots, a_n;$
Case
$s_1 \rightarrow C_1$
$\vdots$
$s_n \rightarrow C_n$
Endcase
D
achieves the goal G from s.
This contradicts the fact that C is minimal w.r.t (G, s) too. Hence, our assumption is incorrect, i.e., s is not a complete state. □

Theorem 2 *Consider a pair (M, S), where S is a set of (possibly incomplete) states, and 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 and complete w.r.t. G and S then M achieves G from S.*

Proof: To prove the Theorem we will show that $\mathcal{U}_M(s)$ is finite, does not end with a_M^F and for all f in G, $\models holds(f, \Phi(\mathcal{U}_M(s), s))$ for every $s \in S$.

It is easy to see that S can be divided in three disjoint subsets

$S_1 = \{s \in S$: every rule in M which is applicable in s is a termination control rule $\}$,

$S_2 = \{s \in S$: there exists a simple control rule r in M which is applicable in $s\}$, and

$S_3 = S \setminus (S_1 \cup S_2)$.

Because of M is complete and sound w.r.t G and S, $S_3 = \emptyset$.

It is easy to see that for each $s \in S_1$, all rules **if** LHS **then** RHS which are applicable in s are termination rules. Hence, $\mathcal{U}_M(s) = []$.
Furthermore, because LHS satisfies G (Definition 7.8), the empty plan achieves G from s. Thus, for all f in G, $\models holds(f, \Phi(\mathcal{U}_M(s), s))$ (1).

Also, because of the empty plan is the minimal cost plan achieving G from a state s if s satisfies G, we can easily conclude that $s \in S_1$ if s satisfies G. (2)

We will now prove that $\mathcal{U}_M(s)$ is finite, does not end with a^F_M, and for all f in G, $\models holds(f, \Phi(\mathcal{U}_M(s), s))$ for $s \in S_2$. There are two cases:

(i) (s is complete): The proof is then similar to the proof of Theorem 1 and therefore is omitted here.

(ii) (s is incomplete): Since M is sound w.r.t. G and S, there exists a minimal cost plan C which achieves G from s and $pref(C) = RHS$.
Because of Lemma 5, if C does not contain a case plan, then $RHS = C$. Thus, for all s' in $\Phi(RHS, s)$, s' satisfies G, i.e., $s' \in S_1$ (because of (2)). Hence, $\mathcal{U}_M(s') = []$ for $s' \in \Phi(RHS, s)$. By definition of $\mathcal{U}_M$, we can conclude that $\mathcal{U}_M(s) = RHS$. Hence, $\mathcal{U}_M(s)$ is finite and for all $f \in G$, $\models holds(f, \Phi(\mathcal{U}_M(s), s))$. (3)
We now need to show that if C contains a case plan, $\mathcal{U}_M(s)$ is finite, does not end with a^F_M, and for all $f \in G$, $\models holds(f, \Phi(\mathcal{U}_M(s), s))$.
Since C is a minimal cost plan w.r.t. (G, s), $RHS = pref(C)$ contains a sensing action (Lemma 5). It implies that for all $s' \in \Phi(RHS, s)$, $s \subset s'$[31].
If $\mathcal{U}_M(s)$ is infinite then there exists a state $s' \in \Phi(RHS, s)$ such that $\mathcal{U}_M(s')$ is infinite. It implies that $s' \in S_2$ too and there exists at least one rule r which is applicable in s' whose RHS contains a sensing action (because (1)-(3)).
Thus, we can conclude that if $\mathcal{U}_M(s)$ is infinite then there exists a sequence of states $k_0 = s, k_1, \ldots, k_t, \ldots$ such that $k_i \in S_2$, $k_i \in \Phi(RHS_{i-1}, k_{i-1})$ and $\mathcal{U}_M(k_i)$ is infinite for every i, $i \geq 0$ where **if** LHS_i **then** RHS_i is a control rule that is applicable in k_i.
Since S_2 is finite, there exists some i and j, $i < j$ such that $k_i = k_j$.
Let $k_i = \langle T_i, F_i \rangle$ for $i = 0, 1, \ldots$

[31] It means that if $s = \langle T, F \rangle$ and $s = \langle T', F' \rangle$ then $T \cup F \subset T' \cup F'$.

From Lemma 4 we have that $T_i \cup F_i \subset T_j \cup F_j$ for $i < j$. This contradicts with our earlier conclusion that $k_i = k_j$.

Hence, $\mathcal{U}_M(s)$ is finite for $s \in S_2$.

Furthermore, we can conclude that if $s' \in \Phi(\mathcal{U}_M(s), s))$ then $s' \in S_1 \cup S_3$. Since $S_3 = \emptyset$, $s' \in S_1$ if $s' \in \Phi(\mathcal{U}_M(s), s))$. Thus, for all $f \in G$ and $s' \in \Phi(\mathcal{U}_M(s), s) \models holds(f, s')$ (from (ii)-(iii)).

This means for all f in G, $\models holds(f, \Phi(\mathcal{U}_M(s), s))$. (4)

The Proposition is proved by (1)-(4). □