Relating theories of actions and reactive robot control

Chitta Baral and Tran Son
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968
chitta@cs.utep.edu

Abstract

In this paper we explore the connection between theories of actions and reactive robot control architectures that are based on the paradigm of situated activity. In particular, we use the entailment relation of the action description language to formalize the notion of 'an action leading to a goal' of Kaelbling and Rosenschein.

Content Areas: Robotics, Theory of Action

Introduction

Research in theories of actions (Bro87; Wor95; Geo94) is concerned with developing formal theories that allows us to represent effects of actions on the world and reason about them. One of the main agenda behind research in reasoning about actions is to develop autonomous agents (robots) that can act in a dynamic world. The early attempts to use theories of reasoning about actions and planning to formulate a robot control architecture were not successful for several reasons:

- The early theories based on STRIPS and its extensions allowed only observations about the initial state. A robot control architecture using these theories was usually of the form: (i) make observations, (ii) use the action theory to construct a plan to achieve the goal, and (iii) execute the plan.

  For such an architecture to work the world must be static so that it does not change during the execution of the plans, and knowledge about relevant conditions must be complete. This assumption is not valid for a dynamic world where other agents may change the world and/or the robot may not have all the information about the environment when it makes the plan.
- Planning is a time consuming activity (ENS92) and it is not usually wise for the robot to spend a lot of time creating a plan, especially when it is supposed to interact with the environment in real time.

This led to the development of several robot control architectures that were reactive in nature (e.g., the approaches in (Fir87; GL87; Bro86) and in the papers in the collection (Mae91)) and usually were based on the paradigm of 'situated activity' which emphasized ongoing physical interaction with the environment as the main aspect in designing autonomous agents. Some advocates of this alternative approach go to the extreme (Bro91) and suggest that reasoning and high level planning may not be at all necessary while some others (Ark91; KR91; Ms91) advocate to combine high level planning and reasoning with situated activity. These approaches were quite successful, especially in the domain of mobile robots.

But unfortunately, the connection between the theories of reasoning about actions and these control architectures has been largely ignored, resulting in less and less interaction between the researchers of the two fields. Some attempts were made in (KR91), but it was not made clear what the formal equivalent of the concept of 'an action leading to a goal' (KR91) was.

¿From an intuitive point of view theories of actions allow us to represent and reason with knowledge about actions and their effects and observations about the world. On the other hand control architectures tell the robot (quickly) what to do in a situation. How are these two related? The action that the control architecture tells the robot to do in a situation must be an action that leads to the goal based on the theory about the actions. Let us consider an analogy with language theory and parsers. The language theory corresponds to the theory of actions and the parsing table and parser correspond to the condition-action 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. The robot in its need to act quickly needs to use condition-action rules (or a similar table). But where do the condition-action rules come from? Currently, they do not come from the corresponding theories of actions, and usually they are hand coded (except in some cases such as (KR91)). We are not going to argue (in this paper) that they should be automatically derived from theories of actions. (Perhaps, they should in many cases.) *We are going to argue that at least we should be able to show that they are cor-*

rect with respect to the theories of actions. In fact, the properties or conditions that guarantee correctness (to be presented later in the paper) can be used as a guide in manually coming up with the control rules. This is the main goal of this paper.

Motivating Example

Let us motivate the main contribution of this paper through an example.

Consider a mobile robot navigating in a building. To make the example simple let us consider the building in Figure 1. 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. Let us now consider a control loop which when executed by a mobile robot will take the robot from the outside of a room to the elevator. We first define the module '*Go_clockwise_one_room*' which takes the robot to the front of the next room in a clockwise manner. We then define a higher level module[1] work where they use called *Goto_elevator* which takes the robot to the front of the elevator.

Module : Go_clockwise_one_room

- **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** *delete_started*
- **if** $\neg$*just_started*, *wall_on_rt* **then** *go_forward*
- **if** $\neg$*just_started*, *corridor_on_rt* **then** *turn_right_90*
- **if** $\neg$*just_started*, *black_door_on_rt* **then** *HALT*

Module : Goto_elevator

- **if** $\neg$*elevator_on_rt* **then** *Go_clockwise_one_room*
- **if** *elevator_on_rt* **then** *HALT*

The modules *Go_clockwise_one_room* and *Goto_elevator* defined above are similar to the programs generated by the Gapps compiler (KR91), and also similar to production systems and active database rules (with a slightly different semantics though). The intuitive semantics of these modules is that in each cycle the condition part of the rules are evaluated and all rules whose conditions are satisfied are fired (i.e., their action part is performed). Of course we assume that the rules are given such that the action part of two rules that may fire simultaneously do not contradict. This loop continues until one of the action is HALT, after which the execution of the loop terminates.

[1]The approach of having several levels of control modules is similar to Firby's work in University of Chicago where three such levels are used.

The first question we would like to answer in this paper is: Is there any connection between these modules and specification of actions and their effects that are used in reasoning about actions?

For example, let us consider the various actions that appear in the above rules and their effects. We represent them as causal rules in the syntax of $\mathcal{A}$ (GL93). (The rules and their meaning are intuitive. Due to lack of space we do not present their formal meaning here.) *It should be noted that the causal rules and the world model in the action theory is from the robot's perspective. It is not from the perspective of an impartial observer. The later is usually used in action theories, but causes problems when mapping sensor values to the world model.*

Causal Rules describing actions in module Goto_elevator

- *Go_clockwise_one_room* **causes** *at_room*$(X+2)$ **if** *at_room*(X), $\neg$*at_room*(349)
- *Go_clockwise_one_room* **causes** *elevator_on_rt* **if** *at_room*(349)
- *Go_clockwise_one_room* **causes** *at_room*(301) **if** *elevator_on_rt*
- *Go_clockwise_one_room* **causes** $\neg$*at_room*(X) **if** *at_room*(X)
- *Go_clockwise_one_room* **causes** $\neg$*elevator_on_rt* **if** *elevator_on_rt*
- The effect of the action *Go_anticlockwise_one_room* can be defined similarly.

Now, what is the connection between the above causal rules and the control module *Goto_elevator*. After the robot executes this control module it has the elevator on its right, i.e., the fluent *elevator_on_rt* becomes true.

Intuitively, based on the sensor readings, the control module directs the robot to take actions that lead to the goal of making *elevator_on_rt* true. *But can we prove the correctness of the control module? And what does it mean when we say an action leads to a goal in a particular situation?* This is where the causal rule plays a role. *Intuitively, it means that, for any set of sensor readings, if the robot constructs a minimal plan (using theories of actions and planning methodologies) to achieve the goal of making elevator_on_rt true, then the first action in that plan is one of the actions (because there may be several minimal plans) that leads to our goal from the situation(s) described by the sensor readings.*

Let us elaborate on this using the *Goto_elevator* control module. Suppose the robot is at room 335. A minimal plan that will take the robot to the elevator consists of executing the action *Go_clockwise_one_room* eight times. The first action of this sequence is of course the action *Go_clockwise_one_room*. Hence,

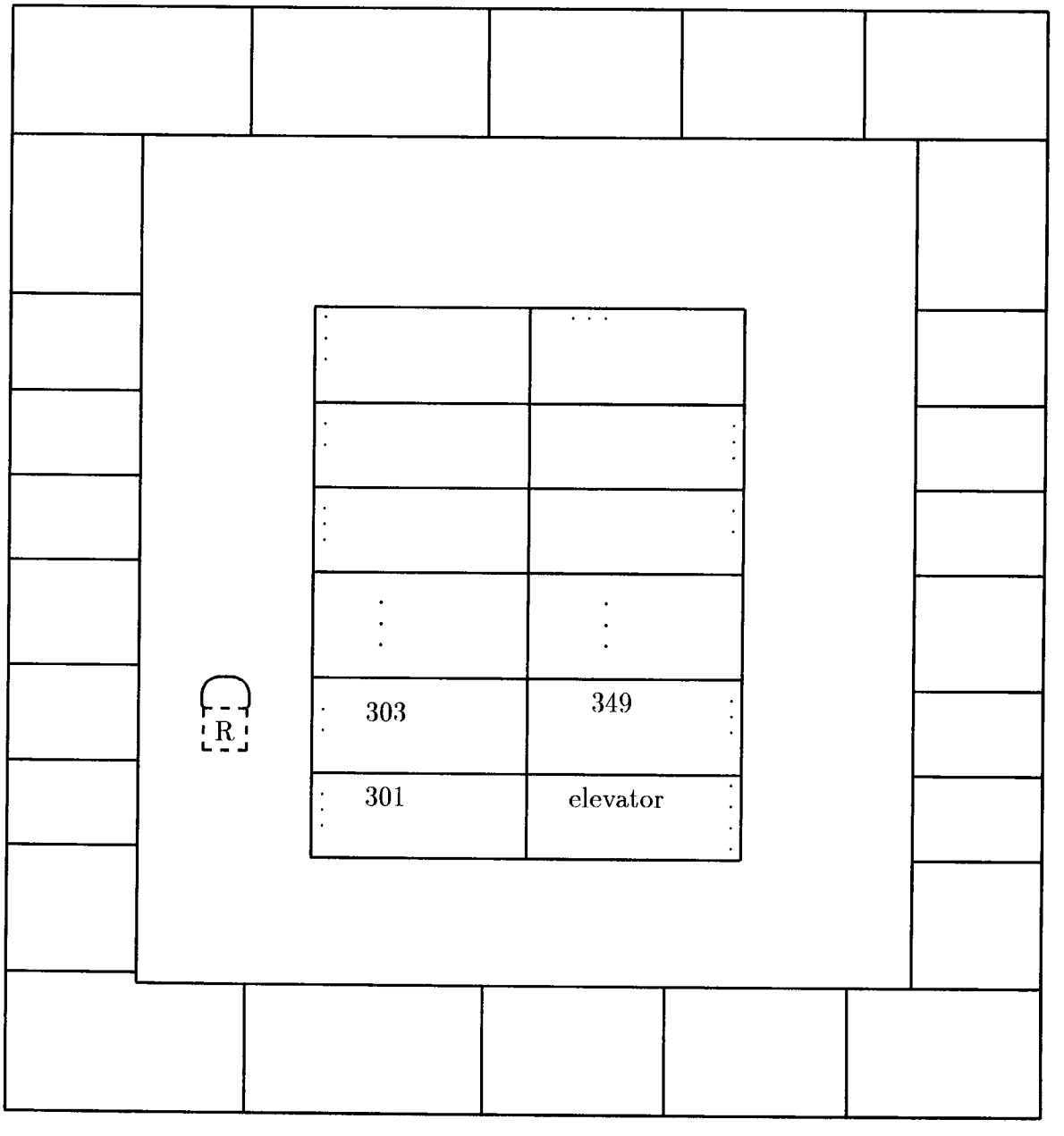

Figure 1:

in the situation where the robot is at room 335 the action that leads to the goal is the action *Go_clockwise_one_room*. The first rule of the control module *Goto_elevator* subsumes this case. Now, let us consider the case when the robot is next to the elevator. Using the causal rules we can conclude that the minimal plan to achieve our goal is the null plan. This is exactly specified by the second rule of *Goto_elevator*. We will formalize the above intuition in the next section. But first we will consider the larger module *Go_clockwise_one_room*, the actions used in it and their causal rules.

Causal Rules describing actions in module *Go_clockwise_one_room*

- *turn_180* **causes** *black_door_on_rt* **if** *white_door_on_rt*
- *turn_180* **causes** $\neg$*white_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** *corrridor_on_rt* **if** *wall_on_rt*, *app_corner*
- *go_forward* **causes** $\neg$*app_corner* **if** *app_corner*
- *turn_right_90* **causes** *wall_on_rt* **if** *corridor_on_rt*
- *delete_started* **causes** $\neg$*just_started* **if** *wall_on_rt*
- **always** (*black_door_on_rt* $\oplus$ *white_door_on_rt* $\oplus$ *wall_on_rt* $\oplus$ *corridor_on_rt*)

Before we discuss the connection between the control module *Go_clockwise_one_room* and the above causal rules, we need to state the effect of executing this module. In this case the effect is that the fluent *just_started* becomes false and the fluent *black_door_on_rt* becomes true. In other words, for the control module to be correct, each rule in the control module *Go_clockwise_one_room* specifies what action leads to the goal $\neg$*just_started* $\wedge$ *black_door_on_rt* for the sensor readings satisfying the conditions in the premise of the rules.

Let us now intuitively explain why the given control module is correct with respect to the causal rules. Suppose the sensors tell the robot that the *white_door_on_rt* is true. Then it is easy to see that all minimal plans that will achieve the goal start with the action *turn_180* and this justifies the correctness of the first rule in the module *Go_clockwise_one_room*. This is justified because there exists a plan which achieves the goal, and because none of the **if** parts of the other causal rules are satisfied and, hence, no other action can achieve anything if executed first. Similarly, when *just_started* is true and a sensor detects that *black_door_on_rt* is true, *go_forward* is the action that is in the beginning of all minimal plan that achieves the goal. This justifies the correctness of the second rule in the module *Go_clockwise_one_room*. The correctness of the other rules in the control module can be explained in a similar manner. ***It should be noted that our only concern is what action is in the beginning of a minimal plan. At this point we are not concerned with how to find a minimal plan.***

We hope that we have provided some intuitive idea of how to connect control modules and causal rules. In the next section we will formalize this connection.

Action theory

Surprisingly, a simple action theory without features such as being able to observe (as in $\mathcal{L}$ (BGP96)), or having narratives (MS94; PR93; BGP95), or having knowledge producing actions (SL93) etc, is sufficient for our purpose.

This is because of the fact that the robot 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 rely on its actions. 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 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 use determine if a fluent f is true in the situation $Res(a_n, Res(a_{n-1}, \ldots, Res(a_1, s)\ldots)$.[2] We denote this by $\models holds(f, [a_1, \ldots, a_n]s)$

But, when we refer to a plan that achieves a goal the plan only consists of actions that the robot can perform. The other kind of action is 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 and that subscribes to our notion of two different kinds of actions is suitable for our purpose.

Formalizing Reactive Modules

We are now ready to formalize reactive control modules, and relate them to action theories. But first, we need to formally define what a control module is.

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

Definition 0.1 (Control rules and Control modules) A *simple control rule* is of the form,

$$\textbf{if } p_1, \ldots, p_k \textbf{ then } a_1, \ldots, a_l \qquad (1)$$

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

$$\textbf{if } p_1, \ldots, p_k \textbf{ then } HALT, \qquad (2)$$

and a *suspension control rule* is of the form

$$\textbf{if } p_1, \ldots, p_k \textbf{ then } SUSPEND, \qquad (3)$$

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. □

The operational semantics of a control module is as follows. The 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.) 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. [3] If the RHS of the rule is $HALT$ then the control module reaches the *success-terminated* stage. If the RHS of the rule is $SUSPEND$ then the control module reaches the *suspended* stage. In the suspended stage the sensors are active and any change in the sensor values takes the robot from the *suspended* stage to the *active* stage. If in the match cycle no rule is found whose LHS is matched then the control module reaches the *failure-terminated* stage.

In Section we discussed the control module of the actions *Goto_elevator* and *Go_clockwise_one_room* whose effects were to achieve goals. We will now give an example of a maintainance control module *Avoid_obstacle* (a similar module is discussed in (JF93)) whose purpose is to maintain the goal $\neg exist\_obstacle$.

Module : Avoid_Obstacle

- **if** *exist_obstacle* **then** *go_around_obstacle*
- **if** $\neg exist\_obstacle$ **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.

Before we formally characterize control modules, we define the closure of a set of states with respect to a control module and an action theory.

Definition 0.2 Let S be a set of states, M be a control module and A be an action theory. By $Closure(S, M, A)$ we denote the smallest set of states that satisfy the following conditions:

- $S \subseteq Closure(S, M, A)$.
- If $s \in Closure(S, M, A)$ and a is an action in A that can occur independent of the robot then $Res(a, s) \in Closure(S, M, A)$.[4]
- If $s \in Closure(S, M, A)$ and there exist a rule in M whose LHS is satisfied by s and whose RHS is $a_1, \ldots, a_l$, then $[a_1, \ldots, a_l]s \in Closure(S, M, A)$

Definition 0.3 A set of states S is said to be *closed* w.r.t. a control module M and an action theory A if $S = Closure(S, M, A)$. □

We will now formally characterize the effect of executing a control module. Intuitively, a control module M executed in a state s executes a sequence of actions. For a control module M, an unfold interpretation $\mathcal{U}_M$ is a function from states to set of sequences of actions.

Definition 0.4 For an achievement (or a mixed) control module M, we say $\mathcal{U}_M$ is an *unfold model* if the following conditions are satisfied.

- If $\mathcal{U}_M(s) = []$ then there exists a termination control rule r in M which is applicable in s

[3] 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.

[4] 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$\}$.

- If $\mathcal{U}_M(s) = a_1, \ldots, a_n, \ldots$ then there exists a simple control rule **if** LHS **then** $a_1, \ldots, a_l$, $l \leq n$ in M which is applicable in s and $\mathcal{U}_M([a_1, \ldots, a_l]s) = a_{l+1}, \ldots, a_n \ldots$.
- $\mathcal{U}_M(s) = a_F^M$ if there exists no rule r in M which is applicable in s where, the action a_F^M is a special action in our action theory which denotes that the execution of M fails.

□

We are only interested in control modules which have a unique unfolding model. (Later we describe some conditions on control modules that guarantee a unique unfolding model.)

Definition 0.5 An achievement control module M is said to achieve goal G from a set of states S, if for all $\mathcal{U}_M$ and for all s in S $\mathcal{U}_M(s)$ is finite and does not end with a_F^M and 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 $max_{s \in S}|\mathcal{U}_M(s)| = n$.

In the following, we will specify sufficient conditions for a control module M achieves a goal G. We first define the notion of minimal cost plan. To each pair of states s and s' and an action a we assign a number which is called as the cost of the translation from s to s' by means of a and satisfies the following conditions:

- $cost(s, s', a) = \infty$ if $s' \neq Res(a, s)$, and
- $0 < cost(s, s', a) < \infty$ if $s' = [a]s$.

The *cost* function from a state s to a state s' by means of a plan $Q = a \circ P$ is defined as follows:

$$cost(s, s', Q) = cost(s, [a]s, a) + cost([a]s, s', P)$$

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

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

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

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

Definition 0.6 *[Soundness]*

1. A simple control rule r is said to be *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if one of the following conditions holds:
   (a) r is not applicable in every $s \in S$, or
   (b) 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.
2. A termination control rule r is said to be *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if its LHS satisfies G.
3. A 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).

□

Definition 0.7 *[Completeness]* A 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 whose LHS is satisfied, and $[RHS]s$ is defined. □

Definition 0.8 A achievement control module M is said to be *non-conflicting* 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. □

Proposition 0.1 Consider a pair (M, S), where M is a non-conflicting 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 M has a unique unfolding model modulo S. □

Proposition 0.2 Consider a pair (M, S), where M is a non-conflicting control module 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:(sketch)
¿From Proposition 0.1, the control module M has a unique unfolding model. Let us refer to it by $\mathcal{U}_M$. Consider any $s \in S$. The minimal plan requirement in the strongly sound condition guarantees that $\mathcal{U}_M(s)$ is finite. The completeness condition guarantees that the last action in $\mathcal{U}_M(s)$ is not a_F^M. Using induction on the number of cycles the control module has to go through before it stops we can show that for any f in G, the action theory entails $holds(f, \mathcal{U}_M(s))$. □

The conditions in the above proposition may be used as a guide while writing control modules.

It should be noted that in the above formalization of control modules, the dynamic nature of the world is captured by requiring a closed S. An achievement control module only achieves the goal if there is an window where there is no outside interference. This is reasonable because it is impossible for a robot to achieve its goal if it is continuously harassed.

Proposition 0.3 The control module *Goto_elevator* achieves the goal *elevator_on_rt* (with respect to the causal rules of *Goto_elevator*) from the set of states $S = \{\{\neg elevator\_on\_rt, at(301)\}, \ldots, \{\neg elevator\_on\_rt, at(349)\}, \{elevator\_on\_rt\}\}$. □

Although, many of the definitions since Definition 0.4 are with respect to achievement control modules only, the definitions can be easily modified to consider maintainance and mixed modules. In fact the definitions remain largely unchanged when we consider a mixed control module. For maintainance control modules the condition in the first item of Definition 0.4 and in Definition 0.7 is changed to require the RHS to be *SUSPEND*. All other changes are straightforward. With these changes, we can now prove the following results.

Proposition 0.4 The control module *Avoid_obstacle* maintains the goal $\neg exist\_obstacle$ (with respect to the causal rules of *Avoid_obstacle*) from the states $\{\{\neg exist\_obstacle\}, \{exist\_obstacle\}\}$. □

Proposition 0.5 The mixed control module obtained by adding the rules in the control modules *Goto_elevator* and *Avoid_obstacle* and removing the rule with SUSPEND in its RHS from it (i.e. from the union) achieves *elevator_on_rt* and maintains $\neg exist\_obstacle$ (with respect to the causal rules in *Goto_elevator* and *Avoid_obstacle*) from all possible states that can be constructed from the fluents *elevator_on_rt* and *exist_obstacle*. □

Conclusion and Future Work

So far, our formalization of control modules required complete information about the states. But we need to take into account the fact that the sensor values of the robot can not always completely determine the state, and the robot has to decide its action in the presence of incomplete information. This can be partially formalized by modifying Definition 0.8 to characterize when control modules are non-conflicting w.r.t. a set of *incomplete* states. This aspect together with conditional plans that a robot may execute will be further discussed in the full paper.

Other concerns that will be addressed in the full paper are formalization of conditional plans encoded in a control module (ex: Robot is programmed to go to the fridge to get a beer and if it does not find a beer there then to go to the cellar.), formalization of repeatative actions (ex: the module that implements *go_forward*.) etc.

Our work is different from the recently proposed action theories (LLL+94; Kow95; BGP96) that allow observations and other features to reason about and control a robot. Since these theories require on-line planning they can not be the main source of robot-control. However, they can be used as a back-up when the reactive rules fail. We will further elaborate on this in the full paper.

Our work in this paper although related to the work on universal plans (Sch87) has many significant differences. Our control modules are not universal plans. They only achieve the goal from certain (most commonly encountered) states. We believe in general an universal plan will be too big and unmanageable. Also, we consider maintainance goals, which is not considered in (Sch87).

References

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

C. Baral, M. Gelfond, and A. Provetti. Representing Actions I: Laws, Observations and Hypothesis. In *Proc. of AAAI 95 Spring Symposium on Extending Theories of Action: Formal theory and practical applications*, 1995.

C. Baral, M. Gelfond, and A. Provetti. Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming (to appear)*, 1996.

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

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

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

K. Erol, D. Nau, and V.S. Subrahmanian. On the complexity of domain-independent planning. In *AAAI 92*, pages 381–386, 92.

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

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

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

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

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

R. Kowalski. Using metalogic to reconcile reactive with rational agents. MIT Press, 1995.

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

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.

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

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.

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

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.

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

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

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