Formal Theories for Reactive Planning Systems: some considerations raised from an experimental application

Paolo Traverso
IRST - Istituto per la Ricerca Scientifica e Tecnologica
38050 Povo, Trento, Italy

Enrico Giunchiglia
DIST, University of Genova
viale Causa 13, 16145 Genova, Italy

Luca Spalazzi
Istituto di Informatica, University of Ancona
Via Brecce Bianche, 60131 Ancona, Italy

Fausto Giunchiglia
DISA, University of Trento
Via Inama 5, 38100 Trento, Italy
IRST - Istituto per la Ricerca Scientifica e Tecnologica
38050 Povo, Trento, Italy

Abstract

This paper is a first attempt towards a theory of actions for reactive planning systems, i.e. systems able to plan and control execution of plans in a partially known and unpredictable environment. We start from a real world application developed at IRST, discuss some of the fundamental requirements and propose a formal theory based on these requirements. The theory takes into account the following facts: (1) actions may fail, since they correspond to complex programs controlling sensors and actuators which have to work in an unpredictable environment; (2) actions acquire information from the real world by activating sensors and actuators; (3) actions generate and execute plans of actions, since the planner needs to activate different special purpose planners and to execute the resulting plans.

Introduction

A lot of recent research in planning is more and more focusing on *reactive planning systems*, i.e. planning systems which are able to plan and control execution of plans in a partially known and unpredictable environment (see for instance (Beetz and McDermott 1994; Firby 1987; Georgeff and Lansky 1986; Simmons 1990)). While formalizations of classical planners have been proposed (see for instance (Lifschitz 1986) for STRIPS (Fikes and Nilsson 1971)), this is not the case for reactive planning systems. There actually seems to be a big gap between the approaches followed and the issues faced in the implementation of reactive planning systems and theories of actions and planning. For instance, the literature in reactive planning does not mention at all some main theoretical problems, e.g. the frame problem and the ramification problem. Moreover, while most of the theoretical work focus on the problem of "reasoning about actions" at planning time, most of the literature on reactive planning systems is actually concerned with "control of action execution", e.g. execution monitoring, sensors control, interleaving of planning/execution/perception, failure recovering.

This paper should be considered a very preliminary attempt to achieve two main goals (both goals are still far from being completely achieved!). The first goal is to investigate the reasons for this gap, to understand when the gap is only apparent and when it is real, and in the latter case, to understand the requirements in order to bridge it. The second obvious goal is to propose how to bridge this gap with a formal theory which meets the requirements.

The approach we are following in this attempt is somehow unusual. We do not start from a formal theory and explain how it extends existing ones in order to bridge the gap. We go the other way around. First, we start from an experimental application developed at IRST (Antoniol *et al.* 1994b; 1994a; Cimatti *et al.* 1992) (briefly described in Section "The Application"). This application is essentially a reactive system which controls a robot navigating in an in-door environment (e.g. a hospital). A planner deals with actions which have to be executed by real sensor/actuator robot controllers.

Second, we propose a formal theory which takes into account the lessons learned from this application (Section "From the application to the theory"). We sketch only the intuitions behind this theory. This is done by explaining actions behaviours in terms of state transitions. This can be the basis for a formal semantics of different formal languages, e.g. languages based on the

situation calculus (McCarthy and Hayes 1969), dynamic and temporal logics (Rosenschein 1981; Harel 1984; Harel *et al.* 1982), action description languages (Gelfond and V.Lifschitz 1993). This allows us to keep the discussion general and independent of the particular formalism which might be chosen. The theory takes into account the following facts: (1) actions may fail, since they correspond to complex programs controlling sensors and actuators which have to work in an unpredictable environment; (2) actions acquire information from the real world by activating sensors and actuators; (3) actions generate and execute plans of actions, since the planner needs to activate different special purpose planners and to execute the resulting plans.

Third, we discuss the relations and differences, in focus, approaches and contents, with respect to existing theories of actions and theoretical approaches to planning (Section "Final considerations and related work").

This work is still limited in at least two respects. First, the theory we propose is strongly motivated by the problems raised in the particular application. Even if we believe that a significant amount of these problems could be generalized, this has not been investigated yet and different applications might raise issues we have not considered. Second, even if we describe actions semantically (as state transitions), we do not provide a formal account of how reasoning about these actions should be performed. See Traverso and Spalazzi [1995] for a logic (based on Process Logic (Harel *et al.* 1982), an extension of Dynamic Logic (Harel 1984)) which deals with the same problems faced in this paper. While Traverso and Spalazzi [1995] focuses on the formal framework, here we focus on the requirements raised from the application and on how actions should be described in terms of state transitions. The formal theory proposed here does not rely on Process Logic.

The Application

We focus on an experimental real world and large scale application developed at IRST by a large team. In this paper, it is described only to the extent needed to explain the motivations for our theory of actions (for a more detailed description see (Antoniol *et al.* 1994b; 1994a; Cimatti *et al.* 1992)). The application aims at the development of a system able to control and coordinate a mobile robot, navigating in unpredictable environments and performing high level tasks, like transportation in hospitals and offices. A simplified version of the architecture is depicted in Figure 1. In the application, users can request the mobile robot to perform desired tasks (see the "user level" in Figure 1). The planner (see the "planning level") is encharged to plan activities in order to perform the requested tasks and to control their execution. Execution is performed by means of modules controlling robot's sensors and actuators (the "sensing and acting level"). Consider the following example. The user requests to transport loads (e.g. food) to a given department (e.g. a food storage). This is a goal for the planner. First, the planner extracts from the "goal-tactic table" (see Section "The planning level") a program (called "tactic"). Second, the program gets executed and activates a "path planner" and a "mission scheduler". The path planner, given a target location, the current position of the robot (contained in the "database of facts") and a topological map of the building, returns a path plan (e.g. the shortest path) to reach the target location. The mission scheduler allocates time for the current task. Third, the planner activates systems at the "sensing and acting level" (see Section "The sensing and acting level"). These systems execute the plan by means of a set of programs, called "behaviours", which activate and control actuators and sensors. A behaviour is for instance the program called "follow-wall(landmark)", which moves the robot along (the wall of) a corridor of the building till a "landmark" (e.g. the end of the corridor, a particular sign on the wall) is reached. This behaviour makes use of data acquired through a sonar and a camera to keep the robot along the wall while it is moving and to detect and avoid obstacles (e.g. trolleys, people) along the way.

In the remaining of this section we describe some of the fundamental characteristics of the sensing and acting level (Section "The sensing and acting level") and of the planning level (Section "The planning level").

The sensing and acting level

The set of behaviours at the sensing and acting level are complex programs. As a matter of fact, in our application behaviours are implemented by thousands of lines of C and assembler code. They are encharged with all the low level control of sensors' and actuators' commands and data. Most operations (e.g. checking the side distance from the wall, checking the presence of an obstacle in front of the robot and moving the robot forward) are executed in parallel. Information is countinuously acquired through sensors in order to decide the commands to be sent to actuators.

Some of the existing behaviours are executed with the main purpose to acquire information from the external environment. We call these

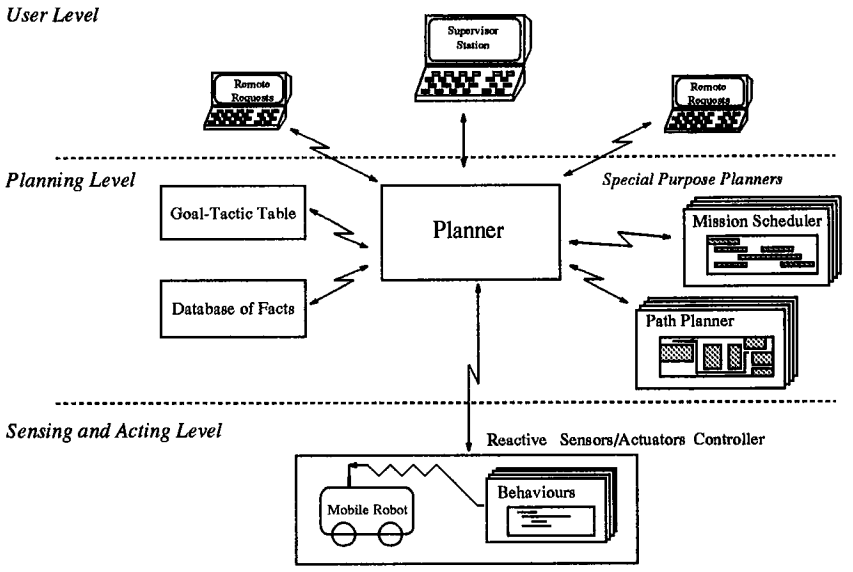

Figure 1: The system architecture

behaviours, *sensing behaviours*. In most cases, sensing behaviours activate actuators in order to put sensors in the position to acquire information. For instance, the behaviour "calibrate(position)" is a complex program which moves the robot and a camera around till the camera can detect a landmark which makes it possible to compute the robot position. This behaviour is necessary in practice. The robot gets often "lost", since the position computed by the actuators controlling the wheels is not reliable when the robot is moving on certain surfaces of the floor.

Behaviours are highly "reactive", i.e. they allow the robot to cope with some of the (many and frequent) unpredictable situations. In our application, it is impossible to predict all possible situations which might arise during execution. For instance, it is impossible to predict whether people moving within the building or trolleys moved around will obstacle the robot's navigation. For this reason, for instance, the behaviour "follow-wall(landmark)" is programmed to avoid unpredictable obstacles along the way.

Nevertheless, behaviours cannot guarantee that their execution will end as expected. Most often, behaviours do not manage to perform their task. For instance, the behaviour "follow-wall(landmark)" might find a not avoidable obstacle along its way. The same behaviour may fail to detect a landmark and get stuck at the end of the corridor. The behaviour "calibrate(position)" may move the camera against an obstacle. All these situations are mainly due to the intrinsic complexity of reality, to the fact that the application domain is unpredictable and highly dynamic and to the fact that actuators and sensors are not perfect (e.g. sonars are not precise enough, wheels cannot follow desired paths precisely). In all these cases, the behaviour is programmed to interrupt execution and report an exception message to the planner. We may think of this as a kind of abort of the program implementing the behaviour. When a behaviour aborts, we say that it *fails* or that there has been a *failure* of the behaviour.

The planning level

The planner processes user requests from the user level. Each user request corresponds to a goal for the planner. For each goal, the planner has a corresponding program which can be executed. We call this kind of programs, *tactics*. Given a goal, the corresponding tactic is retrieved immediately by means of a look-up table (the "goal-tactic table" in Figure 1). At this level, tactics may be thought as "precompiled plans". Actually, tactics are programs which can activate and control the execution of (see Figure 1):

- behaviours at the sensing and acting level, and
- special purpose planners at the planning level.

When a tactic executes a behaviour at the sensing and acting level, it must take into account possible behaviour failures. The tactic traps the abort and reacts to failure. For instance, we have tactics that activate the behaviour "follow-

wall(landmark)" and, when it fails, activate sensing behaviours (e.g. "calibrate(position)") in order to get information about the cause of failure. The behaviour "calibrate(position)" may reveal that the robot is along the expected path. A further sensing behaviour is therefore activated which may detect a not avoidable obstacle along the way. In this case, the tactic activates the path planner to get an alternative path. If "calibrate(position)" reveals the fact that the robot is not along the expected path, the behaviour "follow-wall(landmark)" may have missed to detect the landmark. In this case the tactic lets the robot follow the wall back to the landmark. Notice that tactics manage to "take into account possible failures", but the planner does not try to build plans by "predicting (and thus avoiding) all possible failures". As a matter of fact, in this kind of application domains, behaviour failures cannot always be predicted. In other words, it is impossible that the conditions under which a behaviour may fail can be a priori stated. Notice also that tactics must execute sensing behaviours in order to acquire information about possible causes of failures. A further reason for the need of execution of sensing behaviours is the fact that tactics may need to acquire information which is not available a priori of execution. For instance, most often we cannot predict whether doors are open or closed. The only way to get to know this is to activate sensing behaviours which acquire information at execution time.

The execution of behaviours at the sensing and acting level causes the updating of a database of facts (see Figure 1). In practice, facts are pairs variable-value, expressible by means of atomic propositions. For instance, after that "calibrate(position)" has been executed with success, the database contains the fact *At(position)*, which states the current position of the robot; the successful execution of "follow-wall(landmark)" updates the database by changing the fact *At(position)* with *At(landmark)*.

The system described so far does not do reasoning at all. But some reasoning is actually required. For instance, it is not realistic to hard-code (in a tactic) all the possible paths to reach all the possible target locations. For this reason, tactics can activate "special purpose planners" at the planning level. Special purpose planners generate plans. They are modules which, given in input a goal and some facts, return a tactic. They are constructed to generate plans efficiently and effectively. For instance, the path planner is a dedicated algorithm to search for optimal paths in a topological map. The topological map is a graph whose nodes correspond to locations in the building. Paths are sequences of nodes the robot has to go through. The path planner takes in input particular kinds of goals, i.e. target positions, and information contained in the database of facts about the current position of the robot. It computes a path which gets then translated into a tactic which activates behaviours (at the sensing and acting level) which move the robot along the path. The planning level of the actual application is equipped with a set of very different special purpose planners (the mission scheduler is one of them, see Figure 1).

Some final remarks are in order. First, a tactic can easily interleave the execution of behaviours at the sensing and acting level and of planning activities at the planning level. This is highly required in our application, where it is often necessary to postpone planning activities after some execution is performed which acquires information from the external environment. Second, most often the best way to find the proper plan is simply to ask the user. In our application, the system operator (see Figure 1, user level) can be requested to provide a plan. For instance, when a load is particularly critical or dangerous, the system asks the operator for a path. Finally, the operator might request the planner to give information about the activities of the systems, e.g. the status of the task being performed, the reasons for failures, the information acquired through sensing behaviours, the tasks which have been scheduled and the paths which have been planned.

From the application to the theory

Given the application described so far, we are interested in providing a formal theory for the planner component of the system architecture shown in Figure 1. As an ultimate goal, we aim at a theory of actions which can be used by the planner to reason about actions and thus, to generate plans.

Actions which fail

The theory of actions we aim to must comprise actions which correspond (at the planning level) to behaviours implemented at the sensing and acting level. The activity performed by the planner highly depends on behaviour failures. We have therefore to provide a notion of action failure which corresponds to that of behaviour failure. Notice that a behaviour that fails may modify the world. For instance, "follow-wall(landmark)" and "calibrate(position)" may abort after some navigation has been performed and, as a consequence, the position of the robot (or of the camera) has changed. Even more, if the behaviour is not reliable enough, it may move (or even break) objects

around the robot. As an example independent of our application, think of an action (behaviour) which, while moving a block a from the table on a block b, does not manage to keep the block in hands and the block drops on c. The action has failed, but the action has changed the position of the block.

We consider therefore an action as a transition from an initial state to a final state, where the final state might be "different" from the initial state even in the case the action fails. We call final states where an action α has failed (succeeded), failure (success) states (of the action α). Failure and success of an action α can be formalized with a (kind of) propositional fluent whose argument is the action itself, say $Fail(\alpha)$ and $Succ(\alpha)$. For instance, the action *follow-wall(landmark)* may lead to a state where $Fail($*follow-wall(landmark)*$)$ ($Succ($*follow-wall(landmark)*$)$) holds (see Figure 2). Indeed $Fail($*follow-wall(landmark)*$)$

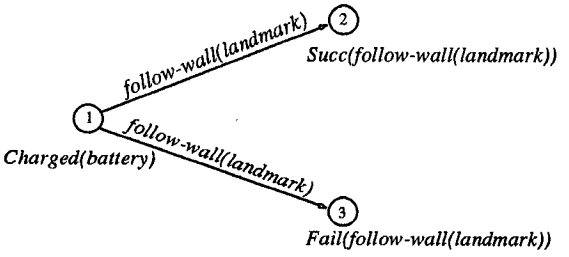

Figure 2: Success and Failure states

and $Succ($*follow-wall(landmark)*$)$ are two atomic propositions which can be contained in the database of facts in Figure 1. This makes the notion of failure very different from that of not executability, which captures the fact that actions are not executable in certain states. Let us suppose that the preconditions for executability of the action *follow-wall(landmark)* are that the battery of the robot is charged, say $Charged($*battery*$)$ (see Figure 2). In state 1, the action is executable, nevertheless it may fail, i.e. end up in state 3, where $Fail($*follow-wall(landmark)*$)$ holds. On the other hand, *follow-wall(landmark)* causes no transitions from states where $\neg Charged($*battery*$)$ holds. From the point of view of failure, behaviours at the sensing and acting level can end up only in two possible ways. Either they terminate without an abort or they abort. We have therefore that in all the "reachable" states of an action α, i.e. the final states of the transition caused by α, either $Fail(\alpha)$ or $Success(\alpha)$ holds. The propositional fluent $Ex(\alpha)$ is defined by

$$Ex(\alpha) \leftrightarrow (Succ(\alpha) \leftrightarrow \neg Fail(\alpha))$$

and holds therefore in any state which is "reachable" by the action α. We have of course states which are not reachable by an action α. For example the state where the block b is picked up is not reachable by the action *follow-wall(landmark)* which is implemented by a behaviour which does not control the robot's manipulator (see Figure 3). As a consequence of this fact, the set of failure states of an action α cannot be defined as the complement of the set of success states of the action α. In other words, $Succ(\alpha) \leftrightarrow \neg Fail(\alpha)$ does not hold in general for any action α.

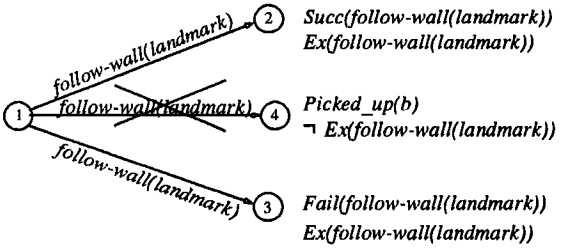

Figure 3: Example of a "not reachable" state

Two further remarks are in order. First, the same state may be a success state for an action α and a failure state for a different action β. This is shown in the example of Figure 4, where we suppose that *landmark1* and *landmark2* are two different landmarks in two different positions of the building.

Second, notice that success/failure of an action does not coincide necessarily with the achievement/not-achievement of a related goal. Intuitively, the former is a property of actions, while the latter is a relation between actions and goals. An action may fail and, nevertheless, achieve the goal the action has been executed for. Indeed, failure of an action corresponds to the fact that the corresponding behaviour aborts. Even if the behaviour aborts, its effects may achieve a desired goal. For example, consider the simple block world example in Figure 5. Let us suppose the planner is given the goal $Clear($c$)$ and, in order to achieve the goal, the planner generates the plan composed of the single action *put-on*(a, b) which moves the block a on the block b. Let us sup-

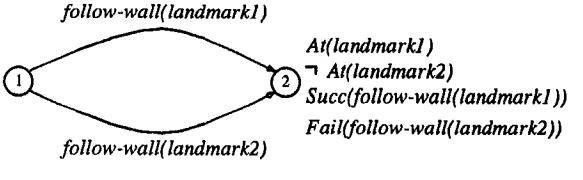

Figure 4: Success state for *follow-wall(landmark1)* and failure state for *follow-wall(landmark2)*

Figure 5: Action which fails [succeeds] and achieves [does not achieve] the goal

pose that the action fails and the block a drops on the table. The action fails and nevertheless achieves the goal *Clear*(c). Vice versa, an action may succeed and may not achieve the goal. Indeed, success of an action corresponds to the fact that the corresponding behaviour does not abort. This does not guarantee that its effects achieve the desired goal. For example, consider again the example in Figure 5. Assume that we have the goal *Far-from*(a,c) for which we plan the action *put-on*(a,table). The action succeeds but it does not achieve the goal for which it has been planned. This is actually what happens in the real application, where sometimes the planner has no choice other than executing actions which may (but are not guaranteed to) achieve desired goals even when they succeed.

The notion of failure described so far allows us to express plans which take into account possible failures and to build a planner able to reason about these plans. The following is a possible example of a plan taking into account failure:

put-on(a, b);
if $Fail$(*put-on*(a, b))
  then {*put-on*(a, table); *put-on*(a, b)}.

In the example, ";" is the usual construct for sequences of actions. The term *if proposition then action* can be regarded as a conditional plan. A tentative formulation in situation calculus of a law of motion and inertia which takes into account failure may be the following:

$$Clear(a, s) \rightarrow Succ(put\text{-}on(a, b), result(put\text{-}on(a, b), s)) \rightarrow On(a, b, result(put\text{-}on(a, b), s)).$$

Actions which sense the world

Notice that all the considerations in Section "Actions which fail" hold for behaviours in general and, in particular, for sensing behaviours. We call actions corresponding to sensing behaviours, *sensing actions*. Thus, for instance, *calibrate(position)* is a sensing action which is executed in order to acquire information, i.e. the position of the robot. As any other action, it may modify the world, e.g. change the position of the camera, and fail. A requirement from the application is that sensing actions can be used in plans. Let us suppose that the behaviour "sense(Closed(door))" activates a camera and a sonar to detect whether a door is open or closed. The following plan takes into account the outcome of the sensing behaviour.

follow–wall(door1);
if Succ(follow-wall(door1)) then {
  sense(Closed(door1));
  if (Succ(sense(Closed(door1))) and
    Closed(door1))
  then open(door1)}.

Sensing actions update the state of knowledge of the system about the world. Consider for instance the action *sense(Closed(door1))*. We introduce the expression *Sensed(Closed(door1))*. (In the application, this expression is contained in the database of facts.) Its intended meaning is "information about the fact whether the door is closed has just been acquired, or in other words, is up-to-date". *Sensed(Closed(door1))* holds therefore if we have just executed *sense(Closed(door1))* with success, i.e. in any final success state of the action *sense(Closed(door1))* (states 2 and 3 in Figure 6). In case of failure, we cannot rely on the fact that the information has been acquired. Therefore $\neg$*Sensed(Closed(door1))* holds in a failure state of the action *sense(Closed(door1))* (see state 4). Moreover, the fact whether we have acquired information with success is independent of the particular result of the sensing action. For this reason, *Sensed(Closed(door1))* holds both in the case *Closed(door1)* and $\neg$*Closed(door1)* hold. Finally, after we execute an action which does not acquire information about the fact whether a door is open or not (e.g. *follow-wall(landmark)*), the information is not up to date, i.e. $\neg$*Sensed(Closed(door1))* holds (see states 5 and 6). Indeed, the last action might change the status of the door, or the door might be moved (closed or opened) by an external agent (e.g. a person). Independently of whether the door is actually closed or opened and of the fact whether it changes status or not, the value of *Closed(door1)* does not change, since the robot cannot realistically get to know this (states 5 and 6). This is what actually happens in the database of facts. The only fact the robot knows is that information about the door has not been "recently" acquired, i.e. $\neg$*Sensed(Closed(door1))*.

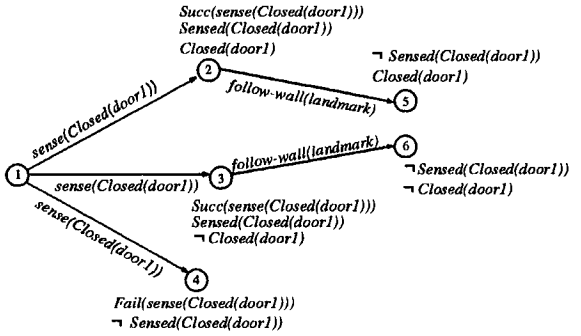

Figure 6: Sensing actions

Actions which generate and execute plans

In Section "The Application" we have discussed some of the different ways the planner can construct a plan. One way is to construct a plan by activating special purpose planners. No matter how the special purpose planners work, they are components which, given goals and facts, return plans ("tactics", in our terminology). In general, plan construction is the execution of some system component code. We therefore represent plan construction as executable actions which construct a plan which can be executed. We extend the language with actions which return a syntactic expression (belonging to the same language) which denotes a plan, i.e. a "name of a plan". We call these actions *plan-for actions*. Intuitively, they construct plans and return names of plans. Plan-for actions are of the form $planfor(\pi, p)$, where p is a proposition (the goal we have to plan for) and π is the name of the plan which is generated. For example, $planfor(\pi, At(landmark))$ can be a plan-for action that invokes the path planner and generates the plan *follow-wall(landmark)* denoted by π.

However, it is not always the case that a plan-for action generates plans by simply activating reasoning modules like special purpose planners. Plan generation may involve the activation of behaviours at the sensing and acting level. For example, the planner may have to ask the user for a plan. In order to capture this extended notion of plan generation, plan-for actions have to be thought simply as actions which construct plans, with no constraints on whether they operate in the real world or not. As a consequence, plan generation may fail in the same way as any other kind of action.

Plan-for actions update the state of knowledge of the system. After that a plan-for action has been executed, the planner has a plan available for execution. We introduce the expression $Planned(\pi, p)$, which holds in any final state of a successful plan-for action which generates a plan denoted by π to achieve the goal p (see Figure 7, state 2).

We require the planner the ability to execute plans denoted by names of plans. We therefore introduce actions of the form $exec(\pi)$ and call them *(plan) execution actions.* Their intended meaning is: "execute the plan denoted by π". Sometimes we write "α" for the name of the plan (action) α. For example, the intended meaning of $exec($"*follow-wall(landmark)*"$)$ is: "execute the plan denoted by "*follow-wall(landmark)*" ".

A reasonable constraint is that the semantics of the execution of a name of an action is the same as that of the action itself. As an example, the state transition of $exec($"*follow-wall(landmark)*"$)$ should be the same as that of *follow-wall(landmark)*. If we assume that executing *follow-wall(landmark)* succeeds but may not achieve the goal, we have the transitions in Figure 7. Finally, notice that plan-for and execution actions allow for interleaving of acting, sensing, planning and execution of a plan. For instance, a possible plan which combines such activities is the following:

follow-wall(landmark);
if Succ(follow-wall(landmark)) then {
  sense(position);
  if Succ(sense(position)) then {
    $planfor(\pi, At(position)); exec(\pi)$}}.

where *sense(position)* is the sensing action which acquires information about the location to be reached.

Final considerations and related work

In the previous sections we have first described an experimental application developed at IRST and then sketched a formal theory which takes into account the lessons learnt from the application. As we already remarked in the introduction, this path is somehow unusual if compared with much of the current research in theories of actions. In theories of action (see for example (Boutilier 1995; Buvac and Costello 1996)) the focus is from the very beginning on how to reason formally about actions and their effects on the world. The hope is that the theory developed will be more or less directly implementable to control a robot or, at least, will serve as a specification for the implementation. Indeed, this does not seem yet to be achievable if the goal is to build a theory and an implementation of robots reasoning, acting and

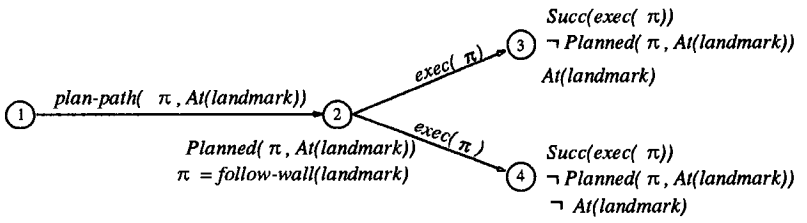

Figure 7: Planning actions

perceiving in a partially and unpredictable environment. As Lesperance *et al.* [1994] acknowledge "[robot programming] remains very tightly coupled to robotic hardware" and even though they feel "that it should be possible to fully control a robot with minimal attention to the details of sensors and effectors, this is certainly not the case today"[1]. Unfortunately, Lesperance's *et al.* commitment of "no implementation without a situation calculus specification" does not seem yet to be exploitable for building robots.

Developing the application described in this paper has not been an exception. Much of the efforts have been in defining a syntax for interfacing the planner with lower level modules (i.e. a syntax for primitive actions and for the interchange of data, e.g. failure). Of course, all the problems we have been dealing with are grounded on the particular application. A different project might have raised a different set of problems. However, we believe that the three main issues we have raised in this paper (failure, sensing and plan-for actions) have to be dealt by any robot-planner operating in a partially known environment. Relaxing this assumption, one might need to introduce less complications in his own formalism and/or implementation. For example, in (Shanahan 1996) a robot-planner is described in which failure and plan-for actions do not seem to play any role. One of the motivations could be the relatively simple robot task (hypothesizing the existence, locations and shapes of objects from a stream of sensor data). Another could be the relatively simple robot's sensors (three bumpers).

Of course, we do believe that it is necessary to provide a formal theory for the application, either a priori or a posteriori. This is what we are trying to do. However, things are very complicate since, as far as we know, few formalizations have been proposed for failure, sensing actions and plan-for actions, and none combining all of them. Even more, some of the traditional issues in theories of actions, like the frame and ramification problems are just behind the corner and await to be faced. Indeed, even though we have never explicitly mentioned the frame problem before, we have to deal with it in our formalization. As a matter of fact, in the previous section we implicitly assumed to have a solution for it, see for example Figure 6. Even though a formal treatment of the frame problem has yet to be carried out, things are made easier than it could be expected since we have to deal with "simple" actions and constraints. By "simple" actions we mean actions whose effects can be described by a set of literals. Analogously for the constraints. As a consequence at the formal level, we have not to deal with all the complications caused by disjunctive information (see for example (Myers and Smith 1988; McCain and Turner 1995)), e.g. ramifications are not possible. As a consequence at the application level, the frame problem can be —and indeed it is— solved by simply updating the knowledge base of the scenario.

About the formalization, it seems that much of the work in theories of action sees planning as deduction. The domain and the effects of actions are represented in a logical formalism (e.g. the situation calculus) and then planning problems are dealt asking whether a certain sentence logically follows from the theory (see for example (Green 1969; Lin and Reiter 1994)). Our first goal is not to provide a formal system with a proof theory, but rather a language with semantics capturing the ideas expressed in previous sections. To this extent, the closest work to ours are the "high level action languages" started after Gelfond and Lifschitz [1993] language $\mathcal{A}$.

Finally, as we have already mentioned, there have been some works treating failure, sensing actions and planning actions. About failure, the closest work is that described in (Rao and Georgeff 1991). Rao and Georgeff extend the computation tree logic CTL* introducing explicit notions for failure and success. However, in their logic actions are not explicitly represented and thus it is not clear how, for example, to do action compositions, i.e. formalize tactics.

A formalization for sensing actions has been

[1] In (Lesperance *et al.* 1994), the comma in the quotation is a full stop.

proposed by Scherl and Levesque [1993] and later extended by Bacchus *et al.* [1995]. There are however some differences between Scherl's and Levesque's formalization and ours. The major difference seems to be that they assume that sensing actions cannot affect the world. We do not have such an assumption. As a matter of fact, at least in our application, it would have been impossible to separate the action of "pure sensing" from the acting involved. As we said, sensing for example the actual position of the robot may involve some adjustments, e.g. in the orientation of the robot. On the other hand, our theory has yet to been refined, for example to introduce some notions analogous to Scherl's and Levesque's [1993] **Knows**, **Kwhether** and **Kref**. Bacchus *et al.* [1995] extend Scherl's and Levesque's work introducing probabilities meant to embody the "degree of confidence" in sensors. In our application, all the sensors' data are equally probable or handled with the same degree of confidence. This has not been a choice but determined by the architecture. The fusion of sensors' data —and thus the eventual handling of inconsistency— happens at the lower level.

Planning actions have been dealt by Steel [1994b] (see also (Steel 1994a)). However, Steel regards planning actions as "non operational", i.e. actions which cannot be executed. In our approach, planning actions are "normal" actions which can be executed, can fail and can change the world. For example, a planning action may need to ask the user for a plan.

Acknowledgments

The application (very briefly) described in this paper was developed at IRST. The acting and sensing level has been all developed by the people at IRST working on Vision and Speech Recognition. Several people at IRST worked on the planning and user level. We thank all of them.

References

G. Antoniol, B. Caprile, A. Cimatti, and R. Fiutem. The Mobile Robot of MAIA: Actions and Interactions in a Real-Life Scenario. In *The Biology and Technology of Intelligent Autonomous Agents*, pages 296–311. Springer Verlag, NATO-ASI Series, 1994.

G. Antoniol, B. Caprile, A. Cimatti, R. Fiutem, and G. Lazzari. Experiencing real-life interaction with the experimental platform of MAIA. In *Proceedings of the 1st European Workshop on Human Comfort and Security*, 1994. Held in conjunction with EITC'94.

F. Bacchus, J.Y. Halpern, and H. Levesque. Reasoning about noisy sensors in the situation calculus. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, 1995.

M. Beetz and D. McDermott. Improving Robot Plans During Their Execution. In *Proceedings 2nd International Conference on AI Planning Systems (AIPS-94)*, Chicago, 1994.

C. Boutilier, editor. *Extending Theories of Action: Formal Theories and Practical Applications: Proceedings of the 1995 AAAI Spring Symposium.* AAAI Press, 1995.

S. Buvac and T. Costello, editors. *Commonsense-96: Working Papers of the Third Symposium on Logical Formalizations of Commonsense Reasoning*, 1996.

A. Cimatti, P. Traverso, S. Dalbosco, and A. Armando. Navigation by Combining Reactivity and Planning. In *Proc. Intelligent Vehicles '92*, Detroit, 1992.

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

R. J. Firby. An Investigation into Reactive Planning in Complex Domains. In *Proc. of the 6th National Conference on Artificial Intelligence*, pages 202–206, Seattle, WA, USA, 1987.

M. Gelfond and V.Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

M. Georgeff and A. L. Lansky. Procedural knowledge. *Proc. of IEEE*, 74(10):1383–1398, 1986.

C. Green. Application of theorem proving to problem solving. In *Proc. of the 1st International Joint Conference on Artificial Intelligence*, pages 219–239, 1969.

D. Harel, D. Kozen, and R. Parikh. Process Logic: expressiveness, decidability, completeness. *Journal of computer and system sciences*, 25:144–170, 1982.

D. Harel. Dynamic Logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. D. Reidel Publishing Company, 1984.

Y. Lesperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R.B. Scherl. A Logical Approach to High-Level Robot Programming - A Progress Report. In *Control of the physical world by intelligent systems, working notes of the 1994 AAAI Fall Symp.*, 1994.

V. Lifschitz. On the semantics of strips. In M. Georgeff and A. Lansky, editors, *Reasoning*

about Actions and Plans, Proceedings of the 1986 Workshop, pages 1–9. Morgan Kaufmann Publ. Inc., 1986.

F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4:655–678, 1994.

N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, 1995.

J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 21–63.

K. Myers and D. Smith. The persistence of derived information. In *Proc. of the 7th National Conference on Artificial Intelligence*, pages 496–500, 1988.

A. S. Rao and M. P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In *Proc. KR'91, Principle of Knowledge Representation and Reasoning*, pages 473–484, Cambridge Massachusetts, 1991. Morgan Kaufmann.

S. Rosenschein. Plan synthesis: A logical perspective. In *Proc. of the 7th International Joint Conference on Artificial Intelligence*, pages 331–337, Vancouver, British Columbia, 1981.

R. Scherl and H.J. Levesque. The frame problem and knowledge producing actions. In *Proc. of the 11th National Conference on Artificial Intelligence*, 1993.

M. Shanahan. Robotics and the Common Sense Informatic Situation. In S. Buvac and T. Costello, editors, *Commonsense-96: Working Papers of the Third Symposium on Logical Formalizations of Commonsense Reasoning*, 1996.

R. Simmons. An Architecture for Coordinating Planning, Sensing and Action. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 292–297, 1990.

S. Steel. Action under Uncertainty. *J. Logic and Computation, Special Issue on Action and Processes*, 4(5):777–795, 1994.

S. Steel. Planning to Plan, 1994. Technical Report, Dept Computer Science, University of Essex, Colchester CO4 3SQ, UK.

P. Traverso and L. Spalazzi. A Logic for Acting, Sensing and Planning. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, 1995.