A Dynamic Logic for Acting, Sensing, and Planning

LUCA SPALAZZI, *Istituto di Informatica, University of Ancona, Via Brecce Bianche, 60131 Ancona, Italy.*
E-mail: spalazzi@inform.unian.it

PAOLO TRAVERSO, *IRST - Istituto per la Ricerca Scientifica e Tecnologica, 38050 Povo, Trento, Italy.*
E-mail: leaf@irst.itc.it

Abstract

This paper is a first attempt towards a theory 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 an experimental 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 need to acquire information from the real world by activating sensors and actuators; (3) actions need to generate and execute plans of actions, since the planner needs to activate different special purpose planners and to execute the resulting plans.

Keywords: Dynamic logic, failure, sensing actions, planning actions, theory of actions, reactive systems.

1 Introduction

A lot of recent research in planning is more and more focusing on *reactive planning systems*, i.e. systems which are able to plan and control execution of plans in a partially known and unpredictable environment (see for instance [5, 14, 17, 18, 41]). While formalizations of classical planners have been proposed (see for instance [32], [37] and [27]), 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 works 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, execution monitoring, sensors control, interleaving of planning/execution/perception, failure recovering.

In this paper we analyse some of the characteristics of reactive planners and propose a formal theory which captures some of the basic aspects of reactive planning. The long term goal is twofold. First, we aim at a better understanding of the requirements and behaviours of reactive planning systems which have to work in real applications. The second goal is to provide an initial framework which can be used to specify reactive planners formally, to prove their properties and to design them in a principled way. This paper should be considered a preliminary attempt to achieve these two main goals.

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. We start from an experimental application developed at IRST [1, 2, 11, 48] (briefly described in Section 2). 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 controllers.

We propose then a logical theory which takes into account the lessons learned from this application (Section 3). The theory takes into account two main facts:

(1) Acting and sensing failures: Actions may fail, since they correspond to complex programs which control sensors and actuators which are not perfect and which have to work in an unpredictable environment. This is the case also of actions that acquire information from the real world, the so-called "sensing actions". Indeed, sensing actions acquire information from the real world by activating both sensors and actuators.

(2) Actions which generate and execute plans: Actions can generate and execute plans of actions, since the planner can activate different special purpose planners in order to construct, execute and control the execution of plans. We call actions which generate and execute plans, "planning actions".

On the one hand, the logical theory is still far from capturing *all* the behaviours of reactive planners. On the other hand, in spite of the fact that the theory we have developed is grounded on the particular application, the theory captures two aspects which are characteristics of reactive planners. First, classical planners assume that it is possible to define the conditions sufficient to guarantee successful action attempts. This assumption is not realistic for reactive planners working in real world applications, where no action, even if apparently simple, is guaranteed to succeed. This is mainly due to the intrinsic complexity of reality, to the fact that the external environment is usually incompletely known and unpredictable, and to the fact that actuators, sensors and models of the world are not perfect: neither acting, nor sensing, nor planning is guaranteed to succeed. As a consequence of fact (1), the logic we propose has in its language the basic operations for failure handling and can therefore reason about failure detection and recovery in acting, sensing and planning.

Second, in reactive planners, planning is not necessarily "off-line and *a priori* reasoning" about action preconditions and effects. The notion of planning is extended *w.r.t.* classical planning in several ways: planning (deliberation) and execution (action) are strictly interconnected and apparently indistinguishable; plans are often pre-compiled rather than generated; plans are modified, abandoned and substituted with alternative plans at run time. The idea of reactive planning involves the whole behaviour of the system, including execution. As a consequence of facts (1) and (2), the theory can express plans which contain both actions which act in the real world (like "move the block"), and actions which perform deliberation (like "construct a plan to move the block") and actions which execute constructed plans (like "execute the plan which moves the block"). During the execution of this kind of plans, the planner can therefore interleave flexibly deliberation and action and perform run time decision making.

Facts (1) and (2) are discussed in detail in Sections 3.1 and 3.2. In these sections we first sketch the intuitions behind the theory. This is done by explaining action 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 [35, 25, 33], dynamic and temporal logics [47], action description languages [16]. This allows us to keep the discussion general

and independent of the particular formalism which might be chosen. We then describe a formal language (syntax and semantics) which formalizes such intuitions. The language is an extension of dynamic logic [26].

In Section 4 we present the deductive system. We prove that it is sound and complete and that validity is decidable. We also show how the theory can be used in verifying plans.

Finally, we draw some conclusions, explain the limitations of the current work and discuss the relations and differences, in focus, approaches and contents, with respect to existing theories of actions and theoretical approaches to planning (Section 5).

2 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 (for a more detailed description see [1, 2, 11]). 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,

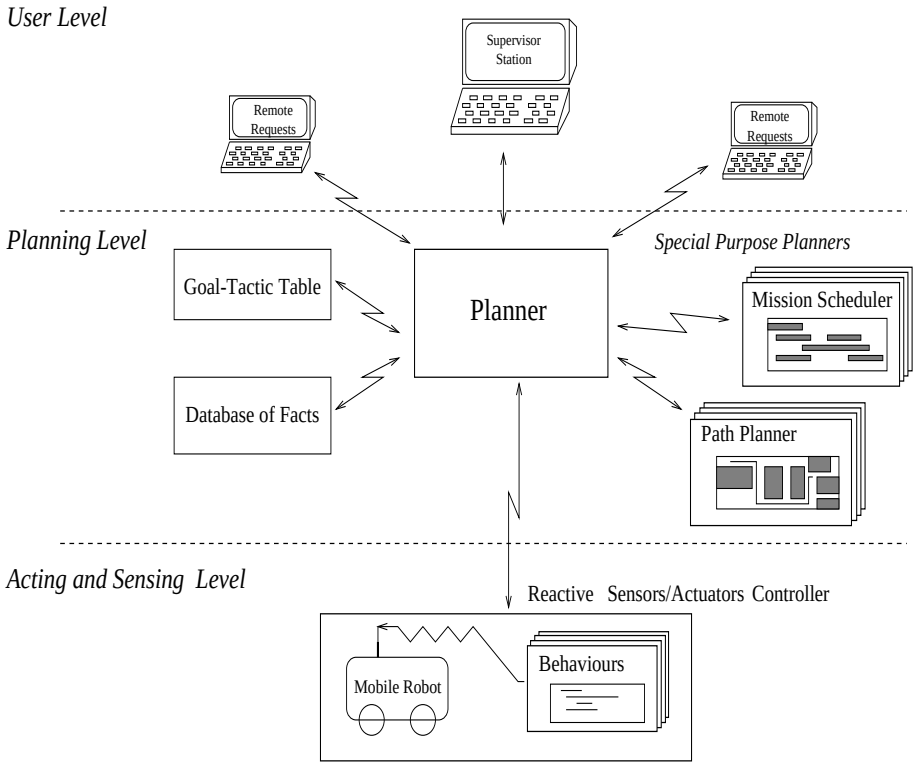

FIGURE 1. The system architecture

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 "acting and sensing 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 2.2) 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 "acting and sensing level" (see Section 2.1). 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 acting and sensing level (Section 2.1) and of the planning level (Section 2.2).

2.1 The acting and sensing level

The set of behaviours at the acting and sensing 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 continuously 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 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 often gets "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 (see Figure 2).

Nevertheless, even if behaviours are reactive, they 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 (see Figure 2). 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

```
void main()
{
   ...
   while (!landmark && !obstacle)
   {
         /* send to the actuator the commmand
            to move the robot of 1 step */
         fprintf(engine,"%d",STEP);

         /* acquire the value of the boolean variable "landmark"
            from the sensor, it is true when the given landmark
            has been detected */
         wait(mutex1);
         fscanf(sensor1, "%h", &landmark);
         signal(mutex1);

         /* acquire the value of the boolean variable "obstacle"
            from the sensor, it is true when the sensor detects
            an obstacle */
         wait(mutex2);
         fscanf(sensor2, "%h", &obstacle);
         signal(mutex2);
   }

   /* if there is an obstacle then abort */
   if (obstacle)
   {
         fprintf(stderr, "There is an obstacle");
         exit(1);
   }

   /* otherwise terminate normally */
   exit(0);
}
```

FIGURE 2. A simplified version of the implementation of the behaviour "follow-wall(landmark)"

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. For example, Figure 2 shows a simplified version of the implementation "follow-wall(landmark)". The program moves the robot (controls the actuators) till the landmark is detected or an obstacle is perceived[1]. When an obstacle is found the program terminates with an exception (the statement `exit(1)`), in other words it aborts. In such a situation we say that the behaviour *fails* or that there has been its *failure*. When it terminates normally (e.g. terminates with `exit(0)`), we say that it *succeeds* or that there has been a *success*. Obviously the system knows how a behaviour is terminated.

[1]The sensors and "follow-wall(landmark)" work concurrently, as a consequence two semaphores to synchronize the behaviours are needed.

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

When a tactic executes a behaviour at the acting and sensing 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 acting and sensing level causes the updating of a database of facts (see Figure 1). In practice, facts are variable-value pairs, 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 hardcode (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 acting and sensing 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 acting and sensing 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, a planner is any module which generates a plan, it does not matter how. The planner may use acting and sensing behaviour, as a consequence it may fail. It may interact with the user as well. 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.

3 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. In section 3.1 we motivate and describe a formal language for describing actions for the acting and sensing level, i.e. actions which might fail and which possibly acquire information from the environment. In section 3.2 we extend the language to represent actions for the planning level, i.e. actions which generate and execute plans.

3.1 Acting and sensing failures

3.1.1 Intuitions

The theory of actions we aim to must comprise actions which correspond (at the planning level) to behaviours implemented at the acting and sensing level. The activity performed by the planner highly depends on behaviour failures. We therefore have 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). Still intuitively, and informally, let us denote the fact that an action α succeeds and the fact that it fails with $Succ(\alpha)$ and $Fail(\alpha)$, respectively. For now, $Fail(\alpha)$ and $Succ(\alpha)$ can be thought as kind of atomic formulas or propositional fluents whose argument is the action itself. For instance, the action *follow-wall(landmark)* may lead to a

state where $Fail$(*follow-wall(landmark)*) ($Succ$(*follow-wall(landmark)*)) holds (see Figure 3). This makes the notion of failure very different from that of not executability, which

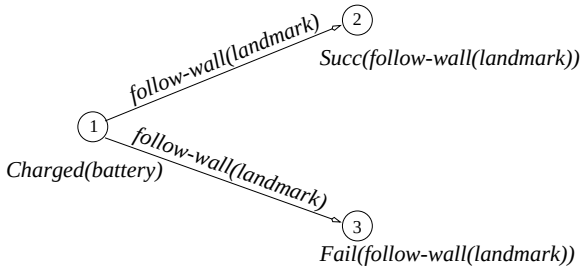

FIGURE 3. Success and Failure states

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 3). 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 the battery is not charged, say $\neg Charged(battery)$.

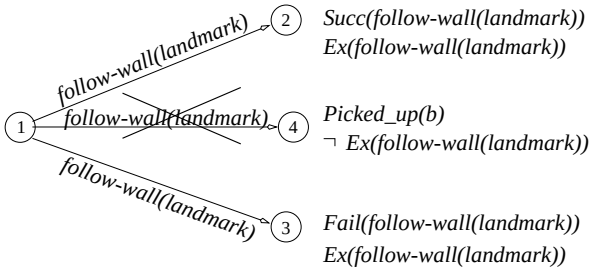

FIGURE 4. Example of a "not reachable" state

¿From the point of view of failure, behaviours at the acting and sensing 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. We denote with $Ex(\alpha)$ the set of states which are "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 behavior which does not control the robot's manipulator (see Figure 4). As a consequence of this fact, the set of failure states of an action α cannot be defined as the complement relative to the entire set of states of the set of success states of the action α. In other words, it is defined as the complement relative to the reachable states of the set of success states.

Some 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 5, where we suppose that *landmark1* and *landmark2* are two different landmarks in two different positions of the building. More in general we may have two actions with the same

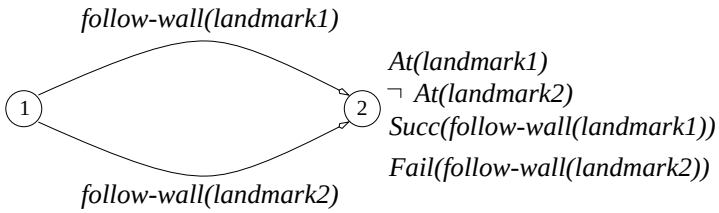

FIGURE 5. Success state for *follow-wall(landmark1)* and failure state for *follow-wall(landmark2)*

behaviour (state transitions) but one succeeds while the other one fails and *vice-versa*. For example, we may have such a situation if we exchange `exit(0)` with `exit(1)` in the C code in Figure 2. 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 code of 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 6. Let us suppose the planner is given the goal $Clear(\mathtt{c})$ and,

FIGURE 6. Action which fails [succeeds] and achieves [does not achieve] the goal

in order to achieve the goal, the planner generates the plan composed of the single action $put\text{-}on(\mathtt{a}, \mathtt{b})$ which moves the block `a` on the block `b`. Let us suppose that the action fails and the block `a` drops on the table. The action fails and nevertheless achieves the goal $Clear(\mathtt{c})$. 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 a system that has the goal of delivering a message to a person, say $Deliver\text{-}to(\mathtt{Fred}, \mathtt{msg1})$. Let us suppose it plans the action $send(\mathtt{mbox1}, \mathtt{msg1})$ since `mbox1` is the usual address of `Fred`. Let us suppose the action succeeds but `Fred` is currently at a different address. 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 not achieve desired goals even when they succeed.

Sensing behaviours are executed to acquire information from the real world through sensors. 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 about the the position of the robot. As any other behaviour, a sensing behaviour may modify the world, e.g. "calibrate(position)" may change the orientation of the robot. As a consequence, at least in our application, it does not seem feasible to separate the action of "pure sensing" from the acting involved. All the considerations about failure in acting hold therefore for sensing actions as well.

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 expression, corresponding to a conditional 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)*}.

The robot first follows the wall to the landmark *door1*. If the action succeeds, the sensing behaviour is activated. The sensing behaviour might fail. If it succeeds and the door is closed, then the robot activates a behaviour which tries to open the door.

Sensing actions update the state of knowledge of the system about the world. Consider for instance the action *sense-Closed(door1)*. If we are in a state where the sensing action has just been executed with success, e.g. *Succ(sense-Closed(door1))* holds, then information about the world has just been acquired, or in other words, is up-to-date, e.g. information about the fact whether the door is closed is up-to-date. Consider for instance figure 7. Information about the status of the door is up-to-date in states 2 and 3. Notice that the fact whether we have acquired information with success is independent of the particular result of the sensing action. For this reason, information is up-to-date, i.e. *Succ(sense-Closed(door1))* holds, both in the case *Closed(door1)* (state 2) and ¬*Closed(door1)* (state 3) hold. In case of failure, we cannot rely on the fact that the information has been acquired. For example, information is not up-to-date in state 4. 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. ¬*Succ(sense-Closed(door1))* holds (see states 5 and 6, where also ¬*Fail(sense-Closed(door1))* hold). 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). Independent 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). The only fact the robot knows is that information about the door has not been "recently" acquired (with success), i.e. ¬*Succ(sense-Closed(door1))*. This notion of sensing actions derive from the application described in Section 2.1. A related and equally important notion is the notion about the knowledge. Briefly, whether the system knows the value of a proposition is different than the proposition value is up to date. The notion of knowledge is not covered by this paper, but it is a future work.

3.1.2 Formal language: Syntax and Semantics

We call the language, $\mathcal{FSP}$ (which stands for Failure, Sensing and Planning). It is an extension of Propositional Dynamic Logic (PDL) [26]. More precisely, it is an extension of Propositional Dynamic Logic with the Converse Operator (CPDL) [26] [2].

The syntax of $\mathcal{FSP}$ is based upon two sets of symbols: $\mathcal{P}_0$, the set of atomic propositions and $\mathcal{A}_0$, the set of atomic actions. From $\mathcal{P}_0$ and $\mathcal{A}_0$ we inductively construct the set $\mathcal{P}$ of

[2] We start from CPDL rather than PDL since CPDL provides some useful technicalities without increasing the complexity of the decidability problem (see for instance [26], pages 538,539). Basically, we need the Converse Operator to define the executability of an action as a state where it is possible to execute the converse action (to go back along the transition graph). This motivation will be much clearer after we will state the condition formally (see condition (3.18) in this section) and give the corresponding axiom (see axiom (REACH) in Section 4.1).

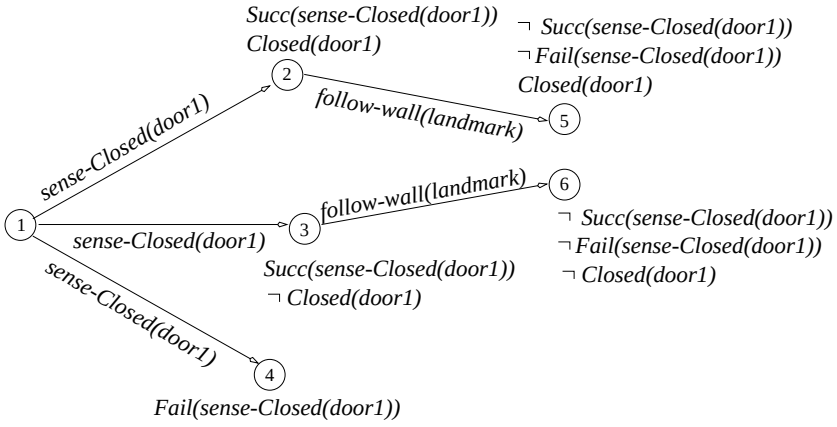

FIGURE 7. Sensing actions

propositions and the set $\mathcal{A}$ of actions. $\mathcal{P}$ and $\mathcal{A}$ include the set of propositions and actions[3] of CPDL, respectively. We refer the reader to [26] for the formal definition of the language of CPDL (a complete definition of the $\mathcal{FSP}$ language, including the CPDL portion, can be found in Appendix A). In order to make this section self-contained, we remember that in PDL, $\mathcal{P}$ is constructed inductively with the usual propositional connectives (e.g. $\wedge$ for conjunction, $\neg$ for negation, $\rightarrow$ for implication, $\vee$ for disjunction); $\mathcal{P}$ includes formulas of the form $[\alpha]p$ (where α is an action and p is a proposition) whose intended meaning is "every possible execution of α leads to a state in which p holds"; compound actions (or plans) in $\mathcal{A}$ are constructed through sequences ($\alpha;\beta$ – do α followed by β), non-deterministic choice ($\alpha \cup \beta$ – do either α or β, non-deterministically), non-deterministic repetition (α^* – repeat α a finite, but non deterministically determined, number of times), and tests ($p?$, where p is a formula – proceed if p is true). CPDL is PDL with the converse operator on actions, inductively allowing α^c as an action in $\mathcal{A}$ for each $\alpha \in \mathcal{A}$ – intuitively, if a possible execution of α starts from a state s and ends in a state t, then (t, s) is a possible execution of α^c. We extend CPDL by inductively allowing the following actions and propositions:

$$\Sigma, \Phi \in \mathcal{A}_0 \tag{3.1}$$

$$\text{If } a \in \mathcal{A}_0, \text{ then } Succ(a), Fail(a) \in \mathcal{P}_0 \tag{3.2}$$

$\mathcal{FSP}$ extends CPDL with the basic constructs which take into account the fact that an action may fail and which provide the capability of controlling failure. Σ and Φ represent the primitive actions that generate success and failure, respectively. Σ (Φ) does nothing but terminate execution with success (failure). $Fail(\alpha)$ and $Succ(\alpha)$ hold iff α has failed and succeeded, respectively. Beside the standard definitions of CPDL (e.g. $\langle\alpha\rangle p$ is an abbreviation of $\neg[\alpha]\neg p$; $\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta$ and $\mathbf{while}\ p\ \mathbf{do}\ \alpha$ are abbreviations of $(p?;\alpha)\cup(\neg p?;\beta)$ and $(p?;\alpha)^*;\neg p?$, respectively), we use the following abbreviations[4]:

$$Ex(\alpha) := Succ(\alpha) \vee Fail(\alpha) \tag{3.3}$$

[3] In [26] actions are called (atomic and compound) programs.

[4] $p := q$ means that the proposition p is an abbreviation for the proposition q. $\alpha := \beta$ means that α is an abbreviation for β. As a consequence we have that $Succ(\alpha) := Succ(\beta)$ and $Fail(\alpha) := Fail(\beta)$.

$$Succ(\alpha;\beta) := Succ(\beta) \wedge \langle(\alpha;\beta)^c\rangle True \tag{3.4}$$

$$Fail(\alpha;\beta) := Fail(\beta) \wedge \langle(\alpha;\beta)^c\rangle True \tag{3.5}$$

$$Succ(\alpha \cup \beta) := Succ(\alpha) \vee Succ(\beta) \tag{3.6}$$

$$Fail(\alpha \cup \beta) := (Fail(\alpha) \wedge Fail(\beta)) \vee (Fail(\alpha) \wedge \neg Ex(\beta)) \vee (\neg Ex(\alpha) \wedge Fail(\beta)) \tag{3.7}$$

$$Succ(\alpha^*) := \neg Fail(\alpha) \tag{3.8}$$

$$Fail(\alpha^*) := Fail(\alpha) \tag{3.9}$$

$$(\Sigma)^c := \Sigma \quad (\Phi)^c := \Phi \tag{3.10}$$

$$\textbf{iffail } \alpha \textbf{ then } \beta \textbf{ else } \gamma := ((\alpha; Fail(\alpha)?); \beta) \cup ((\alpha; Succ(\alpha)?); \gamma) \tag{3.11}$$

$$then(\alpha,\beta) := \textbf{iffail } \alpha \textbf{ then } \Phi \textbf{ else } \beta := ((\alpha; Fail(\alpha)?); \Phi) \cup ((\alpha; Succ(\alpha)?); \beta) \tag{3.12}$$

$$orelse(\alpha,\beta) := \textbf{iffail } \alpha \textbf{ then } \beta \textbf{ else } \Sigma := ((\alpha; Fail(\alpha)?); \beta) \cup ((\alpha; Succ(\alpha)?); \Sigma) \tag{3.13}$$

$$\textbf{repeat}(\alpha) := \alpha; (Succ(\alpha)?; \alpha)^*; (Fail(\alpha)?) \tag{3.14}$$

$Ex(\alpha)$ states that an action has been executed, i.e. it succeeded or failed. The success (failure) of the sequence $\alpha;\beta$ (abbreviations (3.4) and (3.5)) depends on two facts: first, we are at the end of the sequence, i.e. a reachable state of $\alpha;\beta$ (expressed by $\langle(\alpha;\beta)^c\rangle True$[5]); second, the second action of the sequence, i.e. β, succeeds (fails). Consider now abbreviations (3.6) and (3.7). $\alpha \cup \beta$ succeeds when at least one of the two actions succeeds. There are three possible ways in which $\alpha \cup \beta$ can fail. The first case is obvious: both α and β fail. In the other two cases, $\alpha \cup \beta$ fails when only one of the two actions can reach the final state and the other one cannot. This avoids any ambiguity in the case when one of the two actions succeeds and the other one fails. In this case the condition assures that $\alpha \cup \beta$ succeeds. Consider abbreviations (3.8) and (3.9). According to CPDL, α^* stands for n repetitions of α, with $n \geq 0$. As a consequence, a state is the final state of α^* when it is the final state of (one or more concatenations of) α or the final state of no concatenation of α, i.e. any state is the final state of α^*. We suppose that if we have no concatenations of α (i.e. $n = 0$), α^* succeeds and that if we have one or more concatenations of α (i.e. $n \geq 1$), α^* succeeds or fails depending on the success or failure of α itself. Abbreviation (3.10) takes into account the fact that the converse of an action which does nothing is an action which does nothing, i.e. the action itself. Abbreviations (3.11)-(3.14) define the constructs for failure handling. **iffail** is the basic construct for failure handling. The intended meaning of $\textbf{iffail } \alpha \textbf{ then } \beta \textbf{ else } \gamma$ is: "do α, if α fails do β, otherwise do γ". The PDL sequence of actions $\alpha;\beta$ does not take into account failure: the semantics of $\alpha;\beta$ is such that if α is executable, then β is executed anyway, independently of the failure/success of α. For this reason we define the construct $then$. If the first action fails, $then$ does not execute the second action, but simply terminates execution with failure. $then$ captures the behavior of executions of plans by real planners: if the first action fails, the second is not executed and the control is passed to a module for failure recovery. $orelse$ is the construct for failure recovery. $orelse(\alpha,\beta)$ reacts to a failure of α by executing β. $\textbf{repeat}(\alpha)$ [6] controls failure over the repeated execution of actions: it repeats the execution

[5] The reasons of this choice are clearer observing how the reachability condition (Condition (3.18)) is represented by Axiom (REACH) and Theorem (4.13).

[6] $\textbf{repeat}(\alpha)$ should not be confused with the CDPL assertion $repeat(\alpha)$ which states that a program α can

of α till α fails. Intuitively, if α never fails, execution does not terminate. Notice that if it terminates, $\mathbf{repeat}(\alpha)$ always succeeds.

The semantics is based on the standard semantics for CPDL. It is defined relative to a given structure $\mathcal{U}$ of the form $\mathcal{U} = (\mathcal{W}, \rho, \mathcal{R})$, where $\mathcal{W}$ is the set of states; ρ assigns interpretations to propositions: $\rho(p) \subseteq \mathcal{W}$ is the set of states in which p is true (we write $s \models p$ iff $s \in \rho(p)$); $\mathcal{R}$ assigns interpretations to actions: $\mathcal{R}(\alpha) \subseteq \mathcal{W} \times \mathcal{W}$ is the set of pairs of states (s, t) such that one of the possible executions of α can lead from s to t. We refer to [26] for the definition of $\models$ and $\mathcal{R}$ for CPDL (which is also included as part of the semantics given in Appendix A). We extend $\models$ and $\mathcal{R}$ for CPDL to supply meanings the constructs of $\mathcal{FSP}$ which take into account failure:

$$\textit{(null)} \quad \mathcal{R}(\Sigma) = \{(s,s) \mid s \in \mathcal{W}\}; \qquad \mathcal{R}(\Phi) = \{(s,s) \mid s \in \mathcal{W}\} \tag{3.15}$$

$$\textit{(sf-null)} \quad s \models Succ(\Sigma);\ s \not\models Fail(\Sigma); \qquad s \not\models Succ(\Phi);\ s \models Fail(\Phi) \tag{3.16}$$

$$\textit{(sf-test)} \quad s \models Succ(p?) \text{ iff } s \models p; \qquad s \not\models Fail(p?) \tag{3.17}$$

(3.15) states that Σ and Φ do not change the world. Σ always terminates with a success and Φ with a failure, as stated by condition (3.16). Condition (3.17) is a consequence of the fact that a test never fails. Finally, notice that we have no constraints on the success and failure of an atomic action $(a)^c$, i.e. in general formulas such as $Succ(a) \rightarrow Succ(a^c)$ and $Fail(a) \rightarrow Fail(a^c)$ do not hold. This is a consequence of the fact that in $\mathcal{FSP}$, as well as in CPDL, the converse of an atomic action is an atomic action as well. This simplifies the proof of completeness. The converse of non-atomic actions can be translated in linear-time in actions where the converse actions are only the atomic ones (see [26] page 539).

We impose two further semantical conditions. The first condition states that $Succ(\alpha)$ and $Fail(\alpha)$ must hold only in states which are "reachable" by α:

$$\textit{(reachability)} \quad \rho(Succ(a)) \cup \rho(Fail(a)) = \{s \mid \exists t.(t,s) \in \mathcal{R}(a)\} \quad \text{for each } a \in \mathcal{A}_0 \tag{3.18}$$

In condition (3.18), the fact that "there exists an initial state t of the action a such that s is a final state of a" implies that the initial states of a are reachable from the final states of a, in other words, there exists the converse of a, i.e. a^c. Indeed, condition (3.18) could be stated equivalently as $\rho(Succ(a)) \cup \rho(Fail(a)) = \{s \mid \exists t.(s,t) \in \mathcal{R}(a^c)\}$. The second condition is the following:

$$\textit{(non ambiguity)} \quad \rho(Succ(a)) \cap \rho(Fail(a)) = \emptyset \quad \text{for each } a \in \mathcal{A}_0 \tag{3.19}$$

It makes atomic actions "not ambiguous" with respect to failure and success. Actions are not ambiguous (w.r.t. failure and success) if it is always possible to determine whether an action has succeeded or failed. This is a reasonable condition, since this corresponds to the fact that the planner knows whether the execution of (the code of the behaviour corresponding to) the atomic action has terminated with or without an abort. The semantics of the success and failure of compound actions can be derived by means of the formulas (3.4)-(3.9). They produce a set of recursive equations that has been proved to have a solution (the proof is omitted since it is fairly straightforward).

---

execute forever (see [26], pp. 539).

3.2 Actions which generate and execute plans

3.2.1 Intuitions

In our application, the planner can construct a plan by activating special purpose planners, e.g. path-planners and modules reasoning about available resources (see Section 2.2). These special purpose planners, given a goal, construct a plan which can be executed by the planner itself. The theory must capture this ability of the planner. Since a plan is constructed by executing the code implementing a (special purpose) planner, we express plan generation as an action which can be executed. When the action is executed, it constructs a plan which, in turns, can be executed by the planner. The planner needs therefore to "refer to" the plan which has been constructed. In other words, we need names of plans in the language of the planner itself. The theory must have:

1. Actions which construct plans, called *plan generation actions*.
2. Syntactic expressions which denote plans, called *names of plans*.
3. Actions which, given the name of a plan, execute the plan, called *plan execution actions*.

Plan generation actions can be written as actions 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, *plan-for-moving-blocks*$(\pi, Clear(\mathtt{c}))$ can be a plan generation action that invokes a planner for the block world which generates the plan $put\text{-}on(\mathtt{a}, \mathtt{b})$ denoted by π (see the example in Figure 6). Each special purpose planner can therefore have a corresponding plan generation action. For instance, we might have an action which generates the navigation plan of minimal length to reach a given location (for example, *plan-path*$(\pi, At(landmark))$), an action which generates the best navigation plan according to a different cost function (for example, *plan-best-path*$(\pi, At(landmark))$), and so on. Each plan denoted by a name of plan has a corresponding action that may be executed.

Plan execution actions are of the form $exec(\pi)$. Their intended meaning is: "execute the plan denoted by π". For example, if π is the name of the plan corresponding to the action *follow-wall(landmark)*, the intended meaning of $exec(\pi)$ is: "execute *follow-wall(landmark)*" (see Figure 8). Sometimes we write "α" for the name of the plan (action) α. For example, we may have $exec($"*follow-wall(landmark)*"$)$. In our application, plans generated by plan generation actions are all executed by the same execution mechanism of the planner. We assume therefore we have only one plan execution action[7]. According to this assumption, 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)* (see Figure 8).

Notice that a plan generation action can correspond to any module which, given a goal, returns a plan. The module can be a complex system which generates a plan from "scratch" by searching through the space of states or partial plans (like in classical planners, e.g. [13, 37]), which implements probabilistic reasoning (such as in [14]), or which implements a simple "look-up table" which given a goal returns a pre-compiled plan immediately (such as in [17]). As a consequence, the notion of plan generation action is general enough for a variety of possibly different planning mechanisms.

[7] In principle, we might have different ways to execute plans, i.e we might have different plan execution actions. We limit to the case in which we have only one possible execution mechanism, since this is enough to express the behaviours of the planner in our application.

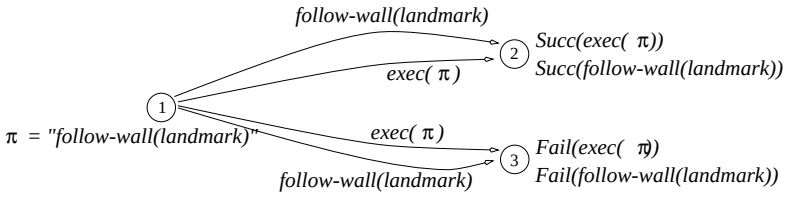

FIGURE 8. Plan execution actions

Given this general notion, a major issue is whether a plan generation action can affect the world. In classical planning the common assumption is that plan generation does not affect the world, i.e. plans are generated by reasoning modules which search for a plan by using an internal model and do not operate in the real world. We do not have such an assumption. As a matter of fact, at least in our application, plan generation may involve the activation of actions at the acting and sensing level. This is also what happens in most of the reactive planners (e.g. in [17, 5]) where, sometimes, the only way to decide for a plan is to do something in the world.

In order to capture this extended notion of plan generation, plan generation 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 generation actions update the state of knowledge of the system. After that a plan generation action has been executed with success, the planner has a plan available for a given goal, or in other words, the plan for that goal is up-to-date. Notice however that, the fact that a plan generation action succeeds in generating a plan does not mean that the execution of the plan will succeed (and neither that it will achieve the goal). This is not realistic in presence of uncertainty. The knowledge of the agent is relative to its (possibly incomplete or wrong) model of the world. Only the execution of the plan will determine whether the plan execution will be successful (and also whether it will achieve the goal or not). As an example, consider Figure 9, where after that the plan *follow-wall(landmark)* (denoted by π) has been generated with success by the action *plan-path*$(\pi, At(landmark))$, the plan (denoted by π) can be executed either with success or with failure.

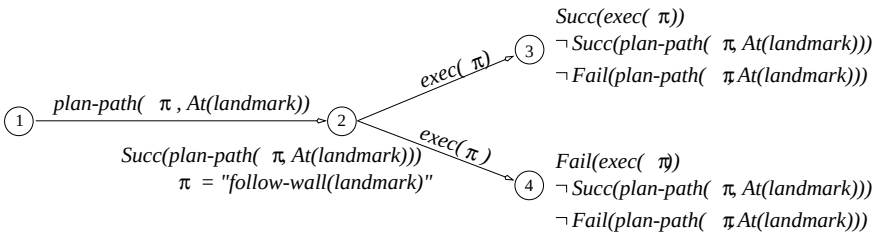

FIGURE 9. Planning actions

Finally, plan generation and execution actions allow for interleaving of planning and execution. For instance, the following example is a plan which decomposes the generation of a plan to reach the location $position_1$ in the generation and execution of a plan to reach an intermediate location in front of the $door_1$, the detection of whether a door is open or closed

and the generation of a second plan followed by the execution of the second plan (see Figure 10).

$\delta =$
then(plan-path(π_1*,*$At(door_1)$*),*
  then(exec(π_1*),*
    then(sense-Closed($door_1$*),*
      if $\neg Closed(door_1)$
      then *then(plan-path(*π_2*,*$At(position_1)$*),exec(*π_2*))*
      else Φ)))

FIGURE 10. An example of plan which interleaves sensing, plan generation, and execution.

3.2.2 Formal language: Syntax and Semantics

We have to extend $\mathcal{FSP}$ to take into account actions that generate and execute plans. In order to manage names of plans we have to introduce the notion of equality among them. This is important in order to allow the planner to decide what to do depending on plans comparisons. This is also a necessary step in order to formalize forms of reasoning like plan transformation and adaptation. Nevertheless we would like to avoid the complexity of a theory with quantification over names of plans. As a consequence we introduce the equality among names of plans as atomic propositions. Furthermore, we introduce the null plan name which does not represent any plan. This is useful to represent when a plan generation does not produce any plan.

Formally, we add to $\mathcal{FSP}$ a set Π of symbols, that we call the set of the *names of plans*. Let Π_0 and $\mathcal{PF}$ be two finite sets of symbols. We inductively define Π as the smallest set such that:

$$\perp \in \Pi,\ \Pi_0 \subseteq \Pi \tag{3.20}$$

$$\text{If } \pi \in \Pi_0,\ p \in \mathcal{P} \text{ and } planfor \in \mathcal{PF}, \text{ then } planfor(\pi, p) \in \mathcal{A}_0 \tag{3.21}$$

$$\text{If } \pi \in \Pi, \text{ then } exec(\pi) \in \mathcal{A}_0 \tag{3.22}$$

$$\text{If } \alpha \in \mathcal{A}, \text{ then } \text{“}\alpha\text{”} \in \Pi \tag{3.23}$$

$$\text{If } \pi_1, \pi_2 \in \Pi, \text{ then } \pi_1 = \pi_2 \in \mathcal{P}_0 \tag{3.24}$$

In (3.21) and (3.22), we extend the set of atomic actions with plan generation and execution actions, respectively. We call actions of the form $planfor(\pi, p)$ and $exec(\pi)$ *plan generation actions* and *plan execution actions*, respectively. We call plan generation and execution actions, *planning actions*. $\perp$ is the null plan name. Π_0 is the set of symbols used to represent plans constructed by plan generation actions. $\mathcal{PF}$ is the set of symbols corresponding to different plan generation mechanisms. Intuitively, we need more than one symbol in Π_0 to represent plans constructed by different (occurrences of) plan generations. For example, given $\Pi_0 = \{\pi_1, \pi_2, \pi_3\}$ and $\mathcal{PF} = \{$*plan-path*, *plan-best-path*$\}$, we might have the following plan

plan-path(π_1,$At(position_1)$);
exec(π_1);
plan-path(π_2,$At(position_2)$);
plan-best-path(π_3,$At(position_2)$);
if $\pi_1 \neq \pi_2 \wedge \pi_2 = \pi_3$ **then** ...

which "stores" in π_1 and π_2 two different occurrences of the plan generated by *plan-path* and in π_3 the plan generated by *plan-best-path* and then compares π_1, π_2 and π_3 in order to decide what to do next.

For any possible plan α in $\mathcal{A}$, we have its name "α" (see (3.23)) . Notice that, since plan generation and execution actions are plans in $\mathcal{A}$, we may have a plan generation action which generates plans composed of plan generation and execution actions. This allows for a kind of "meta-planning" [46]. As an example, the result of the action *meta-plan-path*(π,$At(position_2)$) might be such that

π="*plan-path*(π_1,$At(position_1)$);
exec(π_1);
plan-path(π_2,$At(position_2)$);
plan-best-path(π_3,$At(position_2)$);
if $\pi_1 \neq \pi_2 \wedge \pi_2 = \pi_3$ **then** ..."

In (3.24) we extend the set of atomic propositions with statements expressing the equivalence between names of plans.

The semantics is extend as follows.

(ref) $s \models \pi = \pi$ (3.25)

(sym) $s \models \pi_1 = \pi_2$ iff $s \models \pi_2 = \pi_1$ (3.26)

(tran) $s \models \pi_1 = \pi_2$ and $s \models \pi_2 = \pi_3$ then $s \models \pi_1 = \pi_3$ (3.27)

(nil) $s \not\models$ "α" $= \bot$ (3.28)

(plan) if $s \models Succ(plan for(\pi, p))$ then $s \not\models \pi = \bot$ (3.29)

(exec-i) if $s \models \pi = \bot$ then for all t. $(s,t) \notin \mathcal{R}(exec(\pi))$ (3.30)

(exec-ii) $s \models \pi =$ "α" then for each $p \in \mathcal{P}$
$s \models [exec(\pi)]p \leftrightarrow [\alpha]p$ and
$s \models [exec(\pi)](Succ(exec(\pi)) \leftrightarrow Succ(\alpha)) \wedge$
$(Fail(exec(\pi)) \leftrightarrow Fail(\alpha))$ (3.31)

(3.25), (3.26) and (3.27) are the usual constraints for equivalence[8]. (3.28) and (3.31) are a sort of mapping from names of plans and actions. (3.28) states that a null plan name can not represent any action α, i.e. it can not be equal to the action name "α". Consider (3.31). When a name of plan π represents an action α (i.e. π is equal to the name of α), (3.31) constrains the execution of the plan to behave as α. The intuitive notion that a name of plan can be executed only if it contains an action (i.e. π is not $\bot$) is represented by condition (3.30). (3.29) constrains a successful plan generation to "store" an action in π, i.e. π can not be a null plan name. Notice that, in order to keep the semantics general, the only condition

[8]The equality of names of plans is only syntactical but not semantical. In general, if two actions (α and β) have the same behaviour (i.e. $\mathcal{R}(\alpha) = \mathcal{R}(\beta)$) we can not infer that "α" = "β" holds.

we have given on the interpretations of plan generation actions is condition (3.29). They are atomic actions, and as such they are assigned arbitrary disjoint success and failure sets. Their semantics depends on the particular planning mechanisms which are available to the planning system. As any other atomic action, both plan generation and execution actions have the general restrictions imposed by (3.18) and (3.19). Nevertheless, there are conditions on planning that seem reasonable. We might introduce some of them to further constrain and specify the notion of successful plan generation depending on the domain application and the planning mechanism. We list some of the possible conditions below:

Let $planfor \in \mathcal{PF}$. For each $\pi \in \Pi_0$, $p \in \mathcal{P}$.

(plan-i) $$\rho(Succ(planfor(\pi,p))) \subseteq \Big\{ s \mid \exists t.(s,t) \in \mathcal{R}(exec(\pi)) \text{ and if } t \models Succ(exec(\pi)) \text{ then } t \models p \Big\} \tag{3.32}$$

(plan-ii) $$\rho(Succ(planfor(\pi,p))) \subseteq \Big\{ s \mid \forall t. \text{ if } (s,t) \in \mathcal{R}(exec(\pi)) \text{ and } t \models Succ(exec(\pi)) \text{ then } t \models p \Big\} \tag{3.33}$$

(plan-iii) $$\rho(Succ(planfor(\pi,p))) \subseteq \Big\{ s \mid \exists t.(s,t) \in \mathcal{R}(exec(\pi)) \text{ and } t \models p \Big\} \tag{3.34}$$

(plan-iv) $$\rho(Succ(planfor(\pi,p))) \subseteq \Big\{ s \mid \forall t. \text{ if } (s,t) \in \mathcal{R}(exec(\pi)) \text{ then } t \models p \Big\} \tag{3.35}$$

(plan-v) $$\rho(Ex(planfor(\pi,p))) \subseteq \{ s \mid \exists t.(s,t) \in \mathcal{R}(exec(\pi)) \text{ and } t \models p \} \tag{3.36}$$

(plan-vi) $$\rho(Ex(planfor(\pi,p))) \subseteq \{ s \mid \forall t. \text{ if } (s,t) \in \mathcal{R}(exec(\pi)) \text{ then } t \models p \} \tag{3.37}$$

The conditions (3.32)–(3.37) are different and alternative constraints on the success and failure of planning actions. They constrain the generated plan represented by π to satisfy the goal p under certain conditions. (3.32) allows only for plan generations that when succeed they produce a plan which has at least an execution that if succeeds it achieves the desired goal. (3.33) is a stronger condition, indeed it states that after a successful planning, any successful plan execution satisfies the goal. The conditions (3.34) and (3.35) drop the constraint on the successful plan execution. Any successful plan generation must produce a plan which satisfies the goal in at least one execution (condition (3.34)) or all the executions (condition (3.35)), no matter if they are successes or failures. The conditions (3.36) and (3.37) drop the further constraint on the successful plan generation.

4 The Deductive System

We now introduce an axiom system for $\mathcal{FSP}$ and prove that it is sound and complete and that validity is decidable (Section 4.1). In this system, we can prove theorems about plan failure/success and we can describe actions whose effects depend on their failure/success (Section 4.2). Finally, we give an example that shows how the theory can be used in verifying plans (Section 4.3).

4.1 Axiom System

The deductive system of $\mathcal{FSP}$ is based on the deductive system of CPDL. The axiom system for $\mathcal{FSP}$ includes the Segerberg's axioms, the axioms for the converse actions, and the usual inference rules *modus ponens* and *necessitation* (see [23, 26, 29]). We add the following axiom schemata which take into account failure and planning actions.

For any $a \in \mathcal{A}_0$, $\alpha, \beta \in \mathcal{A}$, $p \in \mathcal{P}$, and $\pi, \pi_1, \pi_2, \pi_3 \in \Pi$:

$$\text{(NULL)} \quad ([\Sigma]p \leftrightarrow p) \wedge ([\Phi]p \leftrightarrow p) \tag{4.1}$$

$$\text{(SF-NULL)} \quad (\neg Fail(\Sigma) \wedge Succ(\Sigma)) \ \wedge \ (\neg Succ(\Phi) \wedge Fail(\Phi)) \tag{4.2}$$

$$\text{(SF-TEST)} \quad (Succ(p?) \leftrightarrow p) \ \wedge \ (\neg Fail(p?)) \tag{4.3}$$

$$\text{(REACH)} \quad Ex(a) \leftrightarrow \langle a^c \rangle True \tag{4.4}$$

$$\text{(NON-AMB)} \quad \neg(Succ(a) \wedge Fail(a)) \tag{4.5}$$

$$\text{(REF)} \quad \pi = \pi \tag{4.6}$$

$$\text{(SYM)} \quad \pi_1 = \pi_2 \leftrightarrow \pi_2 = \pi_1 \tag{4.7}$$

$$\text{(TRAN)} \quad (\pi_1 = \pi_2) \wedge (\pi_2 = \pi_3) \rightarrow (\pi_1 = \pi_3) \tag{4.8}$$

$$\text{(NIL)} \quad \neg(\text{“}\alpha\text{”} = \bot) \tag{4.9}$$

$$\text{(PLAN)} \quad Succ(plan for(\pi, p)) \rightarrow \neg(\pi = \bot) \tag{4.10}$$

$$\text{(EXEC-I)} \quad \pi = \bot \rightarrow \neg\langle exec(\pi) \rangle True \tag{4.11}$$

$$\text{(EXEC-II)} \quad \pi = \text{“}\alpha\text{”} \rightarrow ([exec(\pi)]p \leftrightarrow [\alpha]p) \wedge [exec(\pi)]((Succ(exec(\pi)) \leftrightarrow Succ(\alpha)) \wedge (Fail(exec(\pi)) \leftrightarrow Fail(\alpha))) \tag{4.12}$$

Axioms (4.1)–(4.5) correspond to the semantic restrictions (3.15)–(3.19), axioms (4.6)–(4.12) to restrictions (3.25)–(3.31). Thanks to the possibility of rewriting condition (3.18) as $\rho(Succ(a)) \cup \rho(Fail(a)) = \{s \mid \exists t.(s,t) \in \mathcal{R}(a^c)\}$ we can write the corresponding axiom (REACH). This explains the need for the converse operator mentioned in Section 3.1.

Provability in $\mathcal{FSP}$ is denoted using the $\vdash$ symbol. It is immediate to prove that the conditions (3.18) and (3.19) on atomic actions (corresponding to axioms (REACH) and (NON-AMB)) hold on all actions. Indeed, we can prove in $\mathcal{FSP}$ the following theorems for any $\alpha \in \mathcal{A}$:

$$\vdash Ex(\alpha) \leftrightarrow \langle \alpha^c \rangle True \tag{4.13}$$

$$\vdash \neg(Succ(\alpha) \wedge Fail(\alpha))) \tag{4.14}$$

Axioms (REACH) and (NON-AMB) hold only for atomic actions, Theorems (4.13) and (4.14) state that the above properties can be extended to any action. (4.13) can be reformulated as "$\vdash [\alpha]Ex(\alpha)$", i.e. after the execution of α, the proposition $Ex(\alpha)$ holds.

We prove that $\mathcal{FSP}$ is sound and complete with respect to the previously defined class of structures (see Appendix B).

THEOREM 4.1 (Soundness & Completeness)

For every $p \in \mathcal{P}$

$$\vdash p \text{ iff } \models p$$

We also prove that validity in $\mathcal{FSP}$ is decidable (see Appendix B).

THEOREM 4.2 (Decidability)

Validity in $\mathcal{FSP}$ is decidable.

As mentioned at the end of section 3.2, we may introduce further constraints to plan generation actions. For example, the following axiom schemata correspond to conditions (3.32)–(3.35):

Let $planfor \in \mathcal{PF}$. For each $\pi \in \Pi_0, p \in \mathcal{P}$.

$$\text{(PLAN-I)} \quad Succ(planfor(\pi,p)) \rightarrow \langle exec(\pi)\rangle(Succ(exec(\pi)) \rightarrow p) \tag{4.15}$$

$$\text{(PLAN-II)} \quad Succ(planfor(\pi,p)) \rightarrow [exec(\pi)](Succ(exec(\pi)) \rightarrow p) \tag{4.16}$$

$$\text{(PLAN-III)} \quad Succ(planfor(\pi,p)) \rightarrow \langle exec(\pi)\rangle p \tag{4.17}$$

$$\text{(PLAN-IV)} \quad Succ(planfor(\pi,p)) \rightarrow [exec(\pi)]p \tag{4.18}$$

$$\text{(PLAN-V)} \quad Ex(planfor(\pi,p)) \rightarrow \langle exec(\pi)\rangle p \tag{4.19}$$

$$\text{(PLAN-VI)} \quad Ex(planfor(\pi,p)) \rightarrow [exec(\pi)]p \tag{4.20}$$

It is easy to show that when we introduce one of the above axiom schemata, soundness, completeness, and decidability still hold. These axioms allow us to prove further theorems. For example, it is straightforward to prove from (EXEC-I) and (PLAN-I) the following theorem.

$$Succ(planfor(\pi,p)) \rightarrow \neg\pi = \bot \tag{4.21}$$

Similar theorems are provable from the other axioms.

4.2 Reasoning about failure and success

$\mathcal{FSP}$, like PDL, is a logic of programs. In $\mathcal{FSP}$, like in PDL, we can therefore reason about the effects of actions and plans composed through the usual programming constructs, like conditionals and iterations:

$$\vdash [\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta]q \leftrightarrow (p \rightarrow [\alpha]q) \wedge (\neg p \rightarrow [\beta]q) \tag{4.22}$$

$$\vdash [\mathbf{while}\ p\ \mathbf{do}\ \alpha]q \leftrightarrow (\neg p \rightarrow q) \wedge (p \rightarrow [\alpha][\mathbf{while}\ p\ \mathbf{do}\ \alpha]q) \tag{4.23}$$

Like in PDL, we can describe actions through preconditions and effects. For instance, we might use axioms of the form of (4.24) to describe actions, where p are the preconditions and q the effects of the action α

$$p \rightarrow [\alpha]q \tag{4.24}$$

In $\mathcal{FSP}$ we can reason about failure and success and describe the effects of actions depending on whether they fail or succeed. For instance, in $\mathcal{FSP}$, we can reason about failure and success of plans constructed through conditionals and iterations:

$$\begin{aligned} \vdash\ & \big(\ Succ(\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta) \leftrightarrow (Succ(\alpha) \wedge \langle\alpha^c\rangle p) \vee (Succ(\beta) \wedge \langle\beta^c\rangle\neg p)\ \big) \wedge \\ & \big(\ Fail(\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta) \leftrightarrow (Fail(\alpha) \wedge \langle\alpha^c\rangle p \wedge \neg(Succ(\beta) \wedge \langle\beta^c\rangle\neg p)) \vee \\ & \qquad (Fail(\beta) \wedge \langle\beta^c\rangle\neg p \wedge \neg((Succ(\alpha) \wedge \langle\alpha^c\rangle p)))\ \big) \end{aligned} \tag{4.25}$$

$$\vdash (Succ(\mathbf{while}\ p\ \mathbf{do}\ \alpha) \leftrightarrow \neg p) \ \wedge\ (\neg Fail(\mathbf{while}\ p\ \mathbf{do}\ \alpha)) \tag{4.26}$$

(4.25) states that the success of $\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta$ depends on the success of α when p was true and the success of β otherwise. The case of failure is analogous, with the further condition that we never have the success of an action (e.g. α) with the failure of the other one (e.g. β). Theorem (4.26) states that a **while** loop never fails and ends when the condition (e.g. p) is false.

We can also reason about success and failure of plans containing operators for failure handling. For instance, consider the following theorems:

$$\vdash [\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma]p \leftrightarrow [\alpha]((Fail(\alpha) \rightarrow [\beta]p) \wedge (Succ(\alpha) \rightarrow [\gamma]p)) \tag{4.27}$$

$$\begin{aligned}\vdash\ & \big(Succ(\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma) \leftrightarrow Ex(\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma) \wedge \\ & ((Succ(\beta) \wedge \langle\beta^c\rangle Fail(\alpha)) \vee (Succ(\gamma) \wedge \langle\gamma^c\rangle Succ(\alpha))\big) \wedge \\ & \big(Fail(\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma) \leftrightarrow Ex(\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma) \wedge \\ & ((Fail(\beta) \wedge \langle\beta^c\rangle Fail(\alpha) \wedge \neg Succ(\gamma)) \vee (Fail(\beta) \wedge \langle\beta^c\rangle Fail(\alpha) \wedge \neg\langle\gamma^c\rangle Succ(\alpha)) \vee \\ & (Fail(\gamma) \wedge \langle\gamma^c\rangle Succ(\alpha) \wedge \neg Succ(\beta)) \vee (Fail(\gamma) \wedge \langle\gamma^c\rangle Succ(\alpha) \wedge \neg\langle\beta^c\rangle Fail(\alpha))\big)\end{aligned} \tag{4.28}$$

The theorems above state that when α fails (succeeds), β (γ) is executed and thus the effects, the success and the failure depends on those one of β (γ). The failure also depends on the fact that we do not have a success of γ (β) or γ (β) has not been executed after a success (failure) of α. The proposition $Ex(\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma)$ assures that we are in a state which is reachable by **iffail**. $\mathbf{repeat}(\alpha)$ never fails and ends successfully when α fails:

$$\vdash [\mathbf{repeat}(\alpha)]p \leftrightarrow [\alpha]((Fail(\alpha) \rightarrow p) \wedge (Succ(\alpha) \rightarrow [\mathbf{repeat}(\alpha)]p)) \tag{4.29}$$

$$\vdash (Succ(\mathbf{repeat}(\alpha)) \leftrightarrow Fail(\alpha)) \ \wedge\ (\neg Fail(\mathbf{repeat}(\alpha))) \tag{4.30}$$

Action descriptions of the form (4.24) are often non adequate for reactive systems, where most of the effects of an action will depend on whether it will fail or succeed. In $\mathcal{FSP}$, we can describe actions whose effects depend on the failure/success of the action. Consider for instance (4.31):

$$p \rightarrow [\alpha](Succ(\alpha) \rightarrow q) \tag{4.31}$$

The intuitive semantics of (4.31) is depicted in Figure 11. We have that if the preconditions p

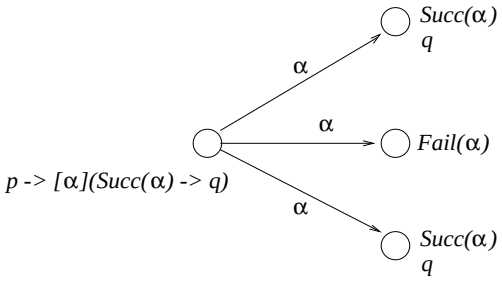

FIGURE 11. Representation of action preconditions and effects with failure

hold before the execution of α, then q holds after its execution only if α succeeds. Notice that success and failure are not predicted, but simply observed. Indeed, in real world applications

there are cases where there is no possibility to predict failure and success. Statements of the kind

$$p \rightarrow [\alpha]Fail(\alpha) \quad p \rightarrow [\alpha]Succ(\alpha) \tag{4.32}$$

can be used to predict failure or success whenever possible.

We can start from action description such as that formalized by (4.31) and reason about the effects of action compositions (see Section 4.3). For instance, the derived rules in figure 12 can be seen as the natural extension of the Hoare's rules [28, 29]. The first rule of figure 12

$$\frac{p_1 \rightarrow [\alpha](Succ(\alpha) \rightarrow q)\ ,\ p_2 \rightarrow [\alpha](Fail(\alpha) \rightarrow r)\ ,\ q \rightarrow [\beta](Succ(\beta) \rightarrow s)\ ,\ r \rightarrow [\beta](Succ(\beta) \rightarrow t)}{(p_1 \wedge p_2) \rightarrow [\alpha;\beta](Succ(\alpha;\beta) \rightarrow s \vee t)}$$

$$\frac{(q_1 \wedge p) \rightarrow [\alpha](Succ(\alpha) \rightarrow r)\ ,\ (q_2 \wedge \neg p) \rightarrow [\beta](Succ(\beta) \rightarrow s)}{(q_1 \wedge q_2) \rightarrow [\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta](Succ(\mathbf{if}\ p\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta) \rightarrow r \vee s)}$$

$$\frac{q \wedge p \rightarrow [\alpha](Succ(\alpha) \rightarrow q)\ ,\ q \wedge p \rightarrow [\alpha](Fail(\alpha) \rightarrow q)}{q \rightarrow [\mathbf{while}\ p\ \mathbf{do}\ \alpha](Succ(\mathbf{while}\ p\ \mathbf{do}\ \alpha) \rightarrow (q \wedge \neg p))}$$

FIGURE 12. Rules for ; **if** and **while**

states that when an action (e.g. α) has the effect of satisfying the precondition of another action (e.g. β), the two actions may be concatenated. The sequence behaves differently depending on the success or failure of α. Notice that, in the third rule of figure 12, the operator **while** needs an invariant (q) that must be true after any execution of α. We can derive another set of inference rules for failure handling operators, e.g. the operators **iffail**, *then*, and **repeat** (see figure 13). The first rule of figure 13 captures the intuitive idea about

$$\frac{p_1 \rightarrow [\alpha](Succ(\alpha) \rightarrow q)\ ,\ p_2 \rightarrow [\alpha](Fail(\alpha) \rightarrow r)\ ,\ q \rightarrow [\gamma](Succ(\gamma) \rightarrow s)\ ,\ r \rightarrow [\beta](Succ(\beta) \rightarrow t)}{(p_1 \wedge p_2) \rightarrow [\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma](Succ(\mathbf{iffail}\ \alpha\ \mathbf{then}\ \beta\ \mathbf{else}\ \gamma) \rightarrow s \vee t)}$$

$$\frac{p \rightarrow [\alpha](Succ(\alpha) \rightarrow q)\ ,\ q \rightarrow [\beta](Succ(\beta) \rightarrow s)}{p \rightarrow [then(\alpha,\beta)](Succ(then(\alpha,\beta)) \rightarrow s)}$$

$$\frac{p \rightarrow [\alpha](Succ(\alpha) \rightarrow p)\ ,\ p \rightarrow [\alpha](Fail(\alpha) \rightarrow p)}{p \rightarrow [\mathbf{repeat}(\alpha)](Succ(\mathbf{repeat}(\alpha)) \rightarrow p)}$$

FIGURE 13. Rules for **iffail**, *then*, and **repeat**

the behaviour of the operator **iffail**. When α fails the overall action behaves as the sequence $\alpha;\beta$. This means that the preconditions of β must be true in the failing state of α. Similar considerations can be made with the preconditions of γ when α succeeds. The second rule formalizes the behaviour of the operator *then*. When an action α has the effect of satisfying the preconditions of another action β and both the actions succeed, the two actions may be concatenated. The difference with the operator ; is that the operator *then* takes into account the failures. Consider the third rule of figure 13. Notice that the operator **repeat** needs an invariant (p) as well as the operator **while**.

4.3 An example

This Section provides an example of plan verification with our theory, i.e. it shows how taking into account failures, sensing, and planning. Let us suppose to be in the situation described

in Section 3.2. The generation of a plan to reach the location $position_1$ is decomposed in the generation and execution of a plan to reach an intermediate location in front of the $door_1$; the detection of wether the door is open or closed; the generation and execution of a second plan. The resulting plan is in Figure 10.

Let us suppose that a plan generation action *plan-path* satisfies the condition (PLAN-II). Consider to have the following laws of motion (that may be axioms or theorems already proved):

$$At(door_1) \rightarrow [\textit{sense-Closed}(door_1)]Succ(\textit{sense-Closed}(door_1)) \tag{4.33}$$

$$Succ(\textit{sense-Closed}(door_1)) \wedge \neg Closed(door_1) \rightarrow [\textit{plan-path}(\pi, At(position_1))]Succ(\textit{plan-path}(\pi, At(position_1))) \tag{4.34}$$

The intuitive meaning of (4.33) is: "when the robot is in front of a door it can successfully sense if the door is open or closed". (4.34) means: "if a robot has successfully executed a sensing action and the door is open then it can successfully generate a plan to go inside". Under these hypothesis we have the following theorem.

THEOREM 4.3
Let δ be the plan defined in Figure 10. Then any successful execution of δ satisfies the goal, i.e.

$$[\delta](Succ(\delta) \rightarrow At(position_1)) \tag{4.35}$$

PROOF. The proof proceeds backward. 4.35 is obtained by means of the *then*-rule (the second rule of Figure 13) from the following formulas.

$$True \rightarrow [\textit{plan-path}(\pi_1, At(door_1))](Succ(\textit{plan-path}(\pi_1, At(door_1))) \rightarrow Succ(\textit{plan-path}(\pi_1, At(door_1)))) \tag{4.36}$$

$$\begin{array}{l} Succ(\textit{plan-path}(\pi_1, At(door_1))) \rightarrow \\ \quad [then(exec(\pi_1), \\ \quad\ then(\textit{sense-Closed}(door_1), \\ \quad\quad \textbf{if } \neg Closed(door_1) \textbf{ then } then(\textit{plan-path}(\pi_2, At(position_1)), exec(\pi_2)) \textbf{ else } \Phi))] \\ \quad (Succ(then(exec(\pi_1), \\ \quad\ then(\textit{sense-Closed}(door_1), \\ \quad\quad \textbf{if } \neg Closed(door_1) \textbf{ then } then(\textit{plan-path}(\pi_2, At(position_1)), exec(\pi_2)) \textbf{ else } \Phi))) \\ \quad\quad \rightarrow At(position_1)) \end{array} \tag{4.37}$$

(4.36) is an easy CPDL theorem[9]. (4.37) can be derived applying the *then*-rule to (4.38) and (4.39).

$$Succ(\textit{plan-path}(\pi_1, At(door_1))) \rightarrow [exec(\pi_1)](Succ(exec(\pi_1)) \rightarrow At(door_1)) \tag{4.38}$$

$$\begin{array}{l} At(door_1) \rightarrow \\ \quad [then(\textit{sense-Closed}(door_1), \\ \quad\quad \textbf{if } \neg Closed(door_1) \textbf{ then } then(\textit{plan-path}(\pi_2, At(position_1)), exec(\pi_2)) \textbf{ else } \Phi)] \\ \quad (Succ(then(\textit{sense-Closed}(door_1), \\ \quad\quad \textbf{if } \neg Closed(door_1) \textbf{ then } then(\textit{plan-path}(\pi_2, At(position_1)), exec(\pi_2)) \textbf{ else } \Phi)) \\ \quad\quad \rightarrow At(position_1)) \end{array} \tag{4.39}$$

(4.38) is the axiom (PLAN-II). Concerning (4.39), it can be obtained if we apply the *then*-rule to (4.40) and (4.41).

$$At(door_1) \rightarrow [\textit{sense-Closed}(door_1)](Succ(\textit{sense-Closed}(door_1)) \rightarrow Succ(\textit{sense-Closed}(door_1))) \tag{4.40}$$

$$\begin{array}{l} Succ(\textit{sense-Closed}(door_1)) \rightarrow \\ \quad [\textbf{if } \neg Closed(door_1) \textbf{ then } then(\textit{plan-path}(\pi_2, At(position_1)), exec(\pi_2)) \textbf{ else } \Phi] \\ \quad\ (Succ(\textbf{if } \neg Closed(door_1) \textbf{ then } then(\textit{plan-path}(\pi_2, At(position_1)), exec(\pi_2)) \textbf{ else } \Phi) \\ \quad\quad \rightarrow At(position_1)) \end{array} \tag{4.41}$$

[9] $True \rightarrow [\alpha](p \rightarrow p)$ for each $\alpha \in \mathcal{A}, p \in \mathcal{P}$.

If we apply *modus ponens* to (4.33) and the following CPDL theorem[10].

$$[\textit{sense-Closed}(door_1)]Succ(\textit{sense-Closed}(door_1)) \rightarrow [\textit{sense-Closed}(door_1)](Succ(\textit{sense-Closed}(door_1)) \rightarrow Succ(\textit{sense-Closed}(door_1))) \tag{4.42}$$

we derive (4.40). We derive (4.41) applying the *if*-rule (the second rule of Figure 12) to (4.43) (that is straightforward to prove) and (4.44).

$$Closed(door_1) \rightarrow [\Phi](Succ(\Phi) \rightarrow At(position_1)) \tag{4.43}$$

$$Succ(\textit{sense-Closed}(door_1)) \wedge \neg Closed(door_1) \rightarrow [then(\textit{plan-path}(\pi_2, At(door_1)), exec(\pi_2))](Succ(then(\textit{plan-path}(\pi_2, At(door_1)), exec(\pi_2))) \rightarrow At(position_1)) \tag{4.44}$$

Again, the *then*-rule can be used to prove (4.44). The formulas to use are (4.45) (the axiom (PLAN-II)) and (4.46).

$$Succ(\textit{plan-path}(\pi_2, At(position_1))) \rightarrow [exec(\pi_2)](Succ(exec(\pi_2)) \rightarrow At(position_1)) \tag{4.45}$$

$$Succ(\textit{sense-Closed}(door_1)) \wedge \neg Closed(door_1) \rightarrow [\textit{plan-path}(\pi_2, At(position_1))](Succ(\textit{plan-path}(\pi_2, At(position_1))) \rightarrow Succ(\textit{plan-path}(\pi_2, At(position_1)))) \tag{4.46}$$

The above formula is proved with *modus ponens*, the axiom (eq-lom-planpath), and the following CPDL theorem[10].

$$[\textit{plan-path}(\pi_2, At(position_1))]Succ(\textit{plan-path}(\pi_2, At(position_1))) \rightarrow [\textit{plan-path}(\pi_2, At(position_1))](Succ(\textit{plan-path}(\pi_2, At(position_1))) \rightarrow Succ(\textit{plan-path}(\pi_2, At(position_1)))) \tag{4.47}$$

∎

5 Final Considerations, Related Work and Future Work

In the previous sections we have first described an experimental application all developed at IRST and then sketched a formal theory which takes into account the lessons learned 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 [6, 7]) 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 perceiving in a partially and unpredictable environment. As Lésperance *et al.* [30] 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"[11]. Unfortunately, Lésperance's *et al.* [30, 31] 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,

[10] $[\alpha]p \rightarrow [\alpha](p \rightarrow p)$ for each $\alpha \in \mathcal{A}, p \in \mathcal{P}$.

[11] In [30], the comma in the quotation is a full stop.

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 main issues we have raised in this paper (failure in acting/sensing and planning 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 [40] a robot-planner is described in which failure and planning 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 complicated since, as far as we know, few formalizations have been proposed for failure, sensing actions and planning 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 Section 3 we implicitly assumed to have a solution for it, see for example Figure 7. Even though a formal treatment of the frame problem has yet to be carried out, in the application described in Section 2 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, as far as we have to formalize applications of this kind, we do not have to deal with all the complications caused by disjunctive information (see for example [36, 34]), 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.

Such a theory can be used to verify system properties and plans. We propose in Section 4.3 an example where it has been proved that a plan satisfies its goal under certain conditions. One of the limitations of our work is that we do not provide any particular proof theory, decision procedure or planning algorithm for building plans. This is one of the major tasks for future research, which can follow two main directions. One possibility is to see planning as deduction. This seem the direction of much of the work in theories of actions and in deductive planning. The domain and the effects of actions are represented in a logical formalism (e.g. the situation calculus, a programming logic) and then planning problems are dealt asking whether a certain sentence logically follows from the theory (see for example [25, 33, 47, 43, 19]). Another possibility would be not to provide a formal system with a proof theory, but rather a language with semantics and a model based decision procedure for planning, like in [8, 9, 10, 22], where planning is performed by model checking within the "high level action language" $\mathcal{AR}$ [20], which extends Gelfond and Lifschitz [16] language $\mathcal{A}$.

There are some works which do not treat failure, sensing actions and planning actions but which are based on the same underlying logic, i.e. a programming logic. See for instance [47, 19]. To this extent, a related work is that of Stephan and Biundo [47], where a tactic based theorem prover is used to generate plans deductively. The constructs we have used for failure handling (e.g. $then$, $orelse$, and **repeat**) have been inspired by the work on tactic-based interactive theorem provers (see for instance [24, 12]).

There have been some works treating failure, sensing actions and planning actions. About failure, the closest work is that described in [38]. 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 plans with the programming logic constructs like sequences, conditionals and iterations.

A formalization for sensing actions has been proposed by Scherl and Levesque [39] and later extended by Bacchus *et al.* [3]. 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 be refined, for example to introduce some notions analogous to Scherl's and Levesque's [39] **Knows**, **Kwhether** and **Kref**. Bacchus *et al.* [3] 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. For some further work on sensing actions see [4].

Planning actions have been dealt by Steel [43] (see also [42]). 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. However, in some recent work Steel considers plan generation and execution as actions of the similar sort [45, 44]. Such actions are formalized using operational semantics instead of a denotational approach, i.e. possible worlds. This approach provides the ability to focus on "how may he do something ..." instead of "what are the effects ...".

Finally, some preliminary ideas and results about this work have been presented in [21, 50, 51, 49]. [21] presents some intuitions about failure, [50, 51] describes a formal framework based on process logic and [49] discusses the intuitions raised from the application described in this paper.

Acknowledgments

The research described in this paper owes a lot to all the members of the Mechanized Reasoning Groups in Trento and Genoa. We thank Fausto Giunchiglia for his invaluable support to the research described in this paper and for all his many conceptual and technical comments on early drafts of this paper. They have contributed to improve the paper significantly. We thank all the people who have participated to the development of the planner for the application described in this paper: in particular we would like to thank Alessandro Cimatti and Roberto Giuri. 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. We would like to thank Enrico Giunchiglia for many fruitful discussions about this work and about the relation with the current research in theories of actions. We also would like to thank Johan van Benthem and Sam Steel for useful discussions about technical contents of this work. Chitta Baral, Frank van Harmelen, Luciano Serafini, Carolyn Talcott, Annette ten Teije and Toby Walsh have provided useful feedback on various aspects

of the work described in this paper. Finally we are thankful to anonymous reviewers for their valuable suggestions.

References

[1] 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. Also as IRST-Technical Report 9606-27 entitled Experimenting Real-Life Interactions with the Mobile Platform of MAIA, IRST, Trento, Italy.

[2] 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. Also IRST-Technical Report 9406-27, IRST, Trento, Italy.

[3] 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.

[4] C. Baral and T.C. Son. Approximate Reasoning About Actions in Presence of Sensing and Incomplete Information. In *Proc. of ILPS-97*, 1997.

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

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

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

[8] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via Model Checking: A Decision Procedure for $\mathcal{AR}$. In *Proceedings of the* 4th European Conference in Planning 1997 (ECP97), Toulouse, France, September 1997.

[9] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proceeding of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, 1998. AAAI-Press.

[10] A. Cimatti, M. Roveri, and P. Traverso. Strong Planning in Non-Deterministic Domains via Model Checking. In *Proceeding of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, Carnegie Mellon University, Pittsburgh, USA, June 1998. AAAI-Press.

[11] A. Cimatti, P. Traverso, S. Dalbosco, and A. Armando. Navigation by Combining Reactivity and Planning. In *Proc. Intelligent Vehicles '92*, Detroit, 1992. IRST-Technical Report 9205-11, IRST, Trento, Italy.

[12] R.L. Constable, S.F. Allen, H.M. Bromley, et al. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice Hall, 1986.

[13] 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.

[14] 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.

[15] M. J. Fischer and R. E. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Science*, 18(2):194–211, 1979. Academic Press.

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

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

[18] M. Ghallab and H. Laruelle. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings 2nd International Conference on AI Planning Systems (AIPS-94)*, Chicago, IL, 1994.

[19] G. De Giacomo, L. Iocchi, and D.N. Rosati. Moving a robot starting from a theory of actions. In *Proceedings of AAAI'96 Workshop on Theories of action, Planning and Control: Bridging the gap*, Portland, Oregon, 1996. AAAI press.

[20] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Representing action: Indeterminacy and ramifications. *Artificial Intelligence*, 1997. To appear.

[21] F. Giunchiglia, L. Spalazzi, and P. Traverso. Planning with Failure. In *Proceedings 2nd International Conference on AI Planning Systems (AIPS-94)*, Chicago, IL, 1994.

[22] F. Giunchiglia and P. Traverso. Planning as Model Checking. In S. Biundo, editor, *Proceedings of the Fifth European Conference on Planning*, Lecture Notes in Artificial Intelligence, Durham, UK, September 1999. Springer-Verlag.

[23] R. Goldblatt. *Logics of Time and Computation.* Number 7 in Lecture Notes. CSLI, Stanford, CA, 2nd edition, 1992.

[24] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanized logic of computation*, volume 78 of *Lecture Notes in Computer Science.* Springer Verlag, 1979.

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

[26] 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.

[27] J. Hertzberg and S. Thiebaux. Turning an Action Formalism into a Planner – A Case Study. *Journal of Logic and Computation*, 4(5):617–654, 1994.

[28] C.A.R. Hoare. An axiomatic basis for computer programming. *Comm. of ACM*, 12:516–580, 1978.

[29] D. Kozen and J. Tiuryn. Logics of Programs. In *Handbook of Theoretical Computer Science*, volume B, pages 789–840. Elsevier, 1990.

[30] Y. Lésperance, 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.

[31] Hector J. Levesque, Raymond Reiter, Ives Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.

[32] 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.

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

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

[35] 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.

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

[37] J. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for adl. In *Proc. of KR-92*, 1992.

[38] 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.

[39] R. Scherl and H.J. Levesque. The Frame Problem and Knowledge Producing Actions. In *Proc. of the 11th National Conference on Artificial Intelligence*, 1993.

[40] 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.

[41] 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.

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

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

[44] S. Steel. A Two-Level View of "Action Logics": with Applications to Cross-Level Planning, 1997. Technical Report, Dept Computer Science, University of Essex, Colchester CO4 3SQ, UK.

[45] S. Steel. Deductive Actions via Operational Semantics Instead of Possible Worlds, 1997. Technical Report, Dept Computer Science, University of Essex, Colchester CO4 3SQ, UK.

[46] M. J. Stefik. Planning and Meta-Planning. *Artificial Intelligence*, 16:141–169, 1981.

[47] W. Stephan and S. Biundo. A New Logical Framework for Deductive Planning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 32–38. Morgan Kaufmann, 1993.

[48] P. Traverso, A. Cimatti, L. Spalazzi, E. Giunchiglia, and A. Armando. MRG: Building planners for real world complex applications. *Applied Artificial Intelligence*, 8(3), 1994. Also IRST-Technical Report 9302-21, IRST, Trento, Italy.

[49] P. Traverso, E. Giunchiglia, L. Spalazzi, and F. Giunchiglia. Formal theories for reactive planning systems: some considerations raised from an experimental application. In *Proceedings of AAAI'96 Workshop on Theories of action, Planning and Control: Bridging the gap*, Portland, Oregon, 1996. AAAI press.

[50] P. Traverso and L. Spalazzi. A Logic for Acting, Sensing and Planning. In *Proc. of the 14th International Joint Conference on Artificial Intelligence,* pages 1941–1949, Montréal, Canada, 1995.

[51] P. Traverso, L. Spalazzi, and F. Giunchiglia. Reasoning about acting, sensing, and failure handling: A logic for agents embedded in the real world. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996.

Appendices

A The Language

($\mathcal{FSP}$)is based on Propositional Dynamic Logic with Converse Operator (CPDL) [26].

Syntax. The syntax of $\mathcal{FSP}$ is based upon four set of symbols: $\mathcal{P}_0$, the denumerable set of atomic propositions, $\mathcal{A}_0$, the denumerable set of atomic actions, $\mathcal{PF}$, the finite set of planning symbols, Π_0, the finite set of plan symbols. ¿From $\mathcal{P}_0$, $\mathcal{A}_0$, $\mathcal{PF}$, and Π_0 we inductively construct the set $\mathcal{P}$ of formulas, the set $\mathcal{A}$ of actions, and the set Π of *action names*.

- If $a \in \mathcal{A}_0$, then $a^c \in \mathcal{A}_0$.
- $\mathcal{A}_0 \subseteq \mathcal{A}$;
- If $\alpha, \beta \in \mathcal{A}$, then $\alpha; \beta \in \mathcal{A}$.
- If $\alpha, \beta \in \mathcal{A}$, then $\alpha \cup \beta \in \mathcal{A}$.
- If $\alpha \in \mathcal{A}$, then $\alpha^* \in \mathcal{A}$.
- If $p \in \mathcal{P}$, then $p? \in \mathcal{A}$.
- $\mathcal{P}_0 \subseteq \mathcal{P}$; $\quad True \in \mathcal{P}$;
- If $p, q \in \mathcal{P}$, then $\neg p$, $p \vee q \in \mathcal{P}$.
- If $p \in \mathcal{P}$, $\alpha \in \mathcal{A}$, then $[\alpha]p \in \mathcal{P}$.
- $\Sigma, \Phi \in \mathcal{A}_0$;
- If $a \in \mathcal{A}_0$, then $Fail(a)$, $Succ(a) \in \mathcal{P}_0$.
- $\perp \in \Pi$, $\Pi_0 \subseteq \Pi$.
- If $planfor \in \mathcal{PF}$, $\pi \in \Pi_0$ and $p \in \mathcal{P}$, then $planfor(\pi, p) \in \mathcal{A}_0$.
- If $\pi \in \Pi$, then $exec(\pi) \in \mathcal{A}_0$;
- If $\alpha \in \mathcal{A}$, then "α" $\in \Pi$.
- If $\pi_1, \pi_2 \in \Pi$, then $\pi_1 = \pi_2 \in \mathcal{P}_0$.

We use $False$, $\wedge$, $\rightarrow$, $\leftrightarrow$ and $\langle\alpha\rangle$ as abbreviations in the standard way. In addition, we use the following abbreviations[12]:

$$Ex(\alpha) := Succ(\alpha) \vee Fail(\alpha) \tag{A.1}$$

$$Succ(\alpha; \beta) := Succ(\beta) \wedge \langle(\alpha; \beta)^c\rangle True \tag{A.2}$$

$$Fail(\alpha; \beta) := Fail(\beta) \wedge \langle(\alpha; \beta)^c\rangle True \tag{A.3}$$

$$Succ(\alpha \cup \beta) := Succ(\alpha) \vee Succ(\beta) \tag{A.4}$$

$$Fail(\alpha \cup \beta) := (Fail(\alpha) \wedge Fail(\beta)) \vee (Fail(\alpha) \wedge \neg Ex(\beta)) \vee (\neg Ex(\alpha) \wedge Fail(\beta)) \tag{A.5}$$

$$Succ(\alpha^*) := \neg Fail(\alpha) \tag{A.6}$$

$$Fail(\alpha^*) := Fail(\alpha) \tag{A.7}$$

[12]Notice that CPDL assumes converse operator is applied to atomic programs only, since Equations (A.8)-(A.10) provide a linear-time translation to the desired formula (see [26], pages 538,539).

$$(\alpha;\beta)^c := \beta^c;\alpha^c \tag{A.8}$$
$$(\alpha\cup\beta)^c := \alpha^c\cup\beta^c \tag{A.9}$$
$$(\alpha^*)^c := (\alpha^c)^* \tag{A.10}$$
$$(\alpha^c)^c := \alpha \tag{A.11}$$
$$(p?)^c := p? \quad (\Sigma)^c := \Sigma \quad (\Phi)^c := \Phi \tag{A.12}$$
$$\textbf{if } p \textbf{ then } \alpha \textbf{ else } \beta := (p?;\alpha)\cup(\neg p?;\beta) \tag{A.13}$$
$$\textbf{while } p \textbf{ do } \alpha := (p?;\alpha)^*;\neg p? \tag{A.14}$$
$$\textbf{iffail } \alpha \textbf{ then } \beta \textbf{ else } \gamma := ((\alpha;Fail(\alpha)?);\beta)\cup((\alpha;Succ(\alpha)?);\gamma) \tag{A.15}$$
$$then(\alpha,\beta) := \textbf{iffail } \alpha \textbf{ then } \Phi \textbf{ else } \beta \tag{A.16}$$
$$:= ((\alpha;Fail(\alpha)?);\Phi)\cup((\alpha;Succ(\alpha)?);\beta)$$
$$orelse(\alpha,\beta) := \textbf{iffail } \alpha \textbf{ then } \beta \textbf{ else } \Sigma \tag{A.17}$$
$$:= ((\alpha;Fail(\alpha)?);\beta)\cup((\alpha;Succ(\alpha)?);\Sigma)$$
$$\textbf{repeat}(\alpha) := \alpha;(Succ(\alpha)?;\alpha)^*;(Fail(\alpha)?) \tag{A.18}$$

Notice that $\mathcal{A}$ is a denumerable set (since it is a set of finite length strings) then Π is denumerable. As a consequence, the relation $=$ which is a subset of $\Pi\times\Pi$ is also denumerable. Finally, $\mathcal{P}$ is also denumerable.

Semantics (Standard Models). The semantics is defined relative to a given structure $\mathcal{U}$ of the form

$$\mathcal{U} = (\mathcal{W},\rho,\mathcal{R})$$

where $\mathcal{W}$ is the set of states, ρ assigns interpretations to propositions (i.e. $\rho(p)\subseteq\mathcal{W}$ is the set of states in which p is true), $\mathcal{R}$ assigns interpretations to actions (i.e. $\mathcal{R}(\alpha)\subseteq\mathcal{W}\times\mathcal{W}$ is the set of pairs which represent the possible executions of α).

$$\rho : \mathcal{P}_0 \to 2^{\mathcal{W}} \qquad \mathcal{R} : \mathcal{A}_0 \to 2^{\mathcal{W}\times\mathcal{W}}$$

We write $s \models p$ (i.e. p is true in s) iff $s\in\rho(p)$. $\models$ and $\mathcal{R}$ are extended inductively to supply meanings for the full sets $\mathcal{P}$ and $\mathcal{A}$ as follows:

1. $\mathcal{R}(\alpha;\beta) = \mathcal{R}(\alpha)\cdot\mathcal{R}(\beta)$
2. $\mathcal{R}(\alpha\cup\beta) = \mathcal{R}(\alpha)\cup\mathcal{R}(\beta)$
3. $\mathcal{R}(\alpha^*) = (\mathcal{R}(\alpha))^*$
4. $\mathcal{R}(p?) = \{(s,s) \mid s\models p\}$
5. $\mathcal{R}(a^c) = \{(s,t) \mid (t,s)\in\mathcal{R}(a)\}$.
6. $\mathcal{R}(\Sigma) = \{(s,s) \mid s\in\mathcal{W}\}$; $\quad\mathcal{R}(\Phi) = \{(s,s) \mid s\in\mathcal{W}\}$.
7. $s \models True$.
8. $s\models\neg p$ iff $s\not\models p$; $\quad s\models p\vee q$ iff $s\models p$ or $s\models q$.
9. $s\models[\alpha]p$ iff $\forall t$. if $(s,t)\in\mathcal{R}(\alpha)$ then $t\models p$.
10. $s\models Succ(p?)$ iff $s\models p$
    $s\not\models Fail(p?)$
11. $s\models Succ(\Sigma)$ $\quad s\not\models Fail(\Sigma)$
    $s\not\models Succ(\Phi)$ $\quad s\models Fail(\Phi)$
12. $s\models Succ(a)\vee Fail(a)$ iff $\exists t.(t,s)\in\mathcal{R}(a)$
13. $s\not\models Succ(a)\wedge Fail(a)$
14. $s\models\pi=\pi$
15. $s\models\pi_1=\pi_2$ iff $s\models\pi_2=\pi_1$
16. if $s\models\pi_1=\pi_2$ and $s\models\pi_2=\pi_3$ then $s\models\pi_1=\pi_3$
17. $s\not\models$ "α" $=\bot$
18. if $s\models Succ(planfor(\pi,p))$ then $s\not\models\pi=\bot$
19. if $s\models\pi=\bot$ then for all t. $(s,t)\notin\mathcal{R}(exec(\pi))$
20. if $s\models\pi=$ "α" then for each $p\in\mathcal{P}$
    - $s\models[exec(\pi)]p\leftrightarrow[\alpha]p$ and

- $s \models [exec(\pi)](Succ(exec(\pi)) \leftrightarrow Succ(\alpha)) \wedge (Fail(exec(\pi)) \leftrightarrow Fail(\alpha))$

Axiom Schemata. Let $\mathcal{FSP}$ be the smallest set of formulas that contains the schemata

Axioms:

(TAUT) all instances of tautologies of the propositional calculus
(K) $[\alpha](p \rightarrow q) \rightarrow ([\alpha]p \rightarrow [\alpha]q)$
(COMP) $[\alpha;\beta]p \leftrightarrow [\alpha][\beta]p$
(ALT) $[\alpha \cup \beta]p \leftrightarrow ([\alpha]p \wedge [\beta]p)$
(MIX) $[\alpha^*]p \leftrightarrow (p \wedge [\alpha][\alpha^*]p)$
(IND) $[\alpha^*](p \rightarrow [\alpha]p) \rightarrow (p \rightarrow [\alpha^*]p)$
(TEST) $[p?]q \leftrightarrow (p \rightarrow q)$
(CONV) $(p \rightarrow [a]\langle a^c\rangle p) \wedge (p \rightarrow [a^c]\langle a\rangle p)$
(REACH) $Ex(a) \leftrightarrow \langle a^c\rangle True$
(NON-AMB) $\neg(Succ(a) \wedge Fail(a))$
(NULL) $([\Sigma]p \leftrightarrow p) \wedge ([\Phi]p \leftrightarrow p)$
(SF-TEST) $(Succ(p?) \leftrightarrow p) \ \wedge \ (\neg Fail(p?))$
(SF-NULL) $(\neg Fail(\Sigma) \wedge Succ(\Sigma)) \ \wedge \ (\neg Succ(\Phi) \wedge Fail(\Phi))$
(REF) $\pi = \pi$
(SYM) $\pi_1 = \pi_2 \leftrightarrow \pi_2 = \pi_1$
(TRAN) $(\pi_1 = \pi_2) \wedge (\pi_2 = \pi_3) \rightarrow (\pi_1 = \pi_3)$
(NIL) $\neg(\text{“}\alpha\text{”} = \bot)$
(PLAN) $Succ(planfor(\pi, p)) \rightarrow \neg(\pi = \bot)$
(EXEC-I) $\pi = \bot \rightarrow \neg\langle exec(\pi)\rangle True$
(EXEC-II) $\pi = \text{“}\alpha\text{”} \rightarrow ([exec(\pi)]p \leftrightarrow [\alpha]p) \wedge [exec(\pi)]((Succ(exec(\pi)) \leftrightarrow Succ(\alpha)) \wedge (Fail(exec(\pi)) \leftrightarrow Fail(\alpha)))$

and it is closed under the following rules

Inference Rules:

(MP) $p, p \rightarrow q \vdash q$ (Modus Ponens)
(NEC) $p \vdash [\alpha]p$ (Necessitation)

B Soundness, Completeness, & Decidability: Proofs

We decompose the theorem 4.1 in two theorems: theorem B.1 and theorem B.13.

THEOREM B.1 (Soundness)

For every $p \in \mathcal{P}$, if $\vdash p$ then $\models p$.

It is immediate to prove from the definition of semantics that all instances of the axioms are valid and that the inference rules preserve validity. In order to prove the theorem B.13 we give in the rest of the section a set of definitions and lemmas.

DEFINITION B.2 (Maximal Set)
A set $s \subseteq \mathcal{P}$ is defined to be maximal if

- s is consistent (i.e. $s \nvdash False$)
- for any $A \in \mathcal{P}$ either $A \in s$ or $\neg A \in s$

LEMMA B.3 (Lindenbaum's Lemma)
Every consistent set of formulas is contained in a maximal set.

See [23] at page 20.

DEFINITION B.4 (Canonical Model)
The canonical model of $\mathcal{FSP}$ is the structure

$$\mathcal{U}^F = (\mathcal{W}^F, \rho^F, \mathcal{R}^F)$$

where
- $\mathcal{W}^F = \{s \subseteq \mathcal{P} \mid s \text{ is maximal}\}$
- $\rho^F(p) = \{s \in \mathcal{W}^F \mid p \in s\}$
- $(s,t) \in \mathcal{R}(\alpha)$ iff $\{A \in \mathcal{P} \mid [\alpha]A \in s\} \subseteq t$ for each $\alpha \in \mathcal{A}$

Notice that $\mathcal{W}^F \neq \emptyset$ is guaranteed by the Lindenbaum's Lemma.

We extend to $\mathcal{FSP}$ the definition of Fischer-Ladner closure [15].

DEFINITION B.5 (Fischer-Ladner closure)

Let $p \in \mathcal{P}$ be a formula. The Fischer-Ladner closure of p, denoted FL(p), is the smallest set S of formulas containing p and satisfying the following closure rules for all $a \in \mathcal{A}_0$, $\alpha, \beta \in \mathcal{A}$, $q, r \in \mathcal{P}$, $\pi_1, \pi_2, \pi_3 \in \Pi$, and $planfor \in \mathcal{PF}$:

$$\begin{aligned}
\neg q \in S &\Rightarrow q \in S \\
q \vee r \in S &\Rightarrow q \in S,\ r \in S \\
[a]q \in S &\Rightarrow q \in S, [a^c]q \in S, Succ(a) \in S,\ Fail(a) \in S \\
[\alpha;\beta]q \in S &\Rightarrow [\alpha][\beta]q \in S \\
[\alpha \cup \beta]q \in S &\Rightarrow [\alpha]q \in S,\ [\beta]q \in S \\
[\alpha^*]q \in S &\Rightarrow q \in S,\ [\alpha][\alpha^*]q \in S \\
[q?]r \in S &\Rightarrow q \in S,\ r \in S, Succ(q?) \in S, Fail(q?) \in S \\
Succ(a) \in S &\Rightarrow [a]True \in S \\
Fail(a) \in S &\Rightarrow [a]True \in S \\
Succ(q?) \in S &\Rightarrow q \in S \\
Fail(q?) \in S &\Rightarrow q \in S \\
&\Rightarrow Succ(\Sigma), Fail(\Sigma) \in S \\
&\Rightarrow Succ(\Phi), Fail(\Phi) \in S \\
\pi_1 = \pi_2 \in S &\Rightarrow \pi_1 = \pi_1 \in S, \pi_2 = \pi_1 \in S \pi_1 = \perp \in S \\
\pi_1 = \pi_2 \in S, \pi_2 = \pi_3 \in S &\Rightarrow \pi_1 = \pi_3 \in S \\
\pi = \text{“}\alpha\text{”} \in S &\Rightarrow [exec(\pi)]True \in S, [\alpha]True \in S \\
[exec(\pi)]q \in S &\Rightarrow \pi = \pi \in S \\
[planfor(\pi, q)]r \in S &\Rightarrow \pi = \pi \in S, q \in S
\end{aligned}$$

LEMMA B.6 (Finiteness of FL Closure)

Let | FL(p) | denote the number of elements in FL(p). Then

| FL(p) | is finite.

PROOF. *Sketch*. The proof proceeds by induction on the structures of actions and formulas. Notice that rules for atomic formulas, $True$ and $False$ do not produce anything. Rules for atomic actions produce formulas. The other rules produce formulas and actions that are shorter and shorter. ∎

Having determined that FL(p) is finite, we perform a filtration.

For each $s \in \mathcal{W}^F$, define

$$FL(p)_s = \{B \in FL(p) \mid \mathcal{U}^F, s \models B\} \in 2^{FL(p)}$$

and put

$$s \sim_{FL(p)} t \quad \text{iff} \quad FL(p)_s = FL(p)_t$$

so that

$$s \sim_{FL(p)} t \text{ iff for all } B \in FL(p), \mathcal{U}^F, s \models B \text{ iff } \mathcal{U}^F, t \models B$$

Then $\sim_{FL(p)}$ is an equivalence relation on $\mathcal{W}^F$. Let

$$| s | = \{t \in \mathcal{W}^F \mid s \sim_{FL(p)} t\}$$

be the $\sim_{FL(p)}$-equivalence class of s, and define

$$\mathcal{W}_{FL(p)} = \{| s | : s \in \mathcal{W}^F\}$$

to be the set of all such equivalence classes.

Now, let $\mathcal{P}0_{FL(p)} = \mathcal{P}_0 \cap FL(p)$ be the set of atomic formulas that belong to $FL(p)$, and define

$$\rho_{FL(p)} : \mathcal{P}0_{FL(p)} \rightarrow 2^{\mathcal{W}_{FL(p)}}$$

by putting

$$| s | \in \rho_{FL(p)}(p) \text{ iff } s \in \rho^F(p) \text{ for any } p \in \mathcal{P}0_{FL(p)}$$

Finally, let $\mathcal{A}_{FL(p)}$ be the smallest set of actions that includes:
all atomic actions occurring in members of $FL(p)$,
all tests $B?$ occurring in members of $FL(p)$
and is closed under ; , $\cup$ and $*$, we can define a FL(p)-Model as follows:

DEFINITION B.7 (FL(p)-Model)
A model

$$\mathcal{U}_{FL(p)} = (\mathcal{W}_{FL(p)}, \rho_{FL(p)}, \mathcal{R}')$$

is defined to be a FL(p)-Model if

- for each atomic action $a \in \mathcal{A}_{FL(p)}$ such that a is not $exec(\pi)$, we use any binary relation such that

$$(| s |, | t |) \in \mathcal{R}'(a) \text{ iff for all } B, \text{ if } [a]B \in FL(p) \text{ and } \mathcal{U}^F, s \models [a]B \text{ then } \mathcal{U}^F, t \models (B \wedge Ex(a) \wedge \neg(Succ(a) \wedge Fail(a))) \quad \text{(B.1)}$$

- for each $exec(\pi) \in \mathcal{A}_{FL(p)}$, we use any binary relation such that

$$\begin{aligned}(| s |, | t |) \in \mathcal{R}'(exec(\pi)) \text{ iff } & \mathcal{U}^F, s \not\models \pi = \bot \\ & \text{and} \\ & \text{for all } B, \text{ for all } \pi = \text{“}\alpha\text{”}, \\ & \quad \text{if } \pi = \text{“}\alpha\text{”} \in FL(p) \text{ and } \mathcal{U}^F, s \models \pi = \text{“}\alpha\text{”} \text{ and} \\ & \qquad [\alpha]B \in FL(p) \text{ and } \mathcal{U}^F, s \models [\alpha]B \\ & \quad \text{then } \mathcal{U}^F, t \models (B \wedge Ex(\alpha) \wedge \neg(Succ(\alpha) \wedge Fail(\alpha))) \\ & \qquad \text{and} \\ & \quad \text{if } [exec(\pi)]B \in FL(p) \text{ and } \mathcal{U}^F, s \models [exec(\pi)]B \\ & \quad \text{then } \mathcal{U}^F, t \models (B \wedge Ex(exec(\pi)) \wedge \\ & \qquad \neg(Succ(exec(\pi)) \wedge Fail(exec(\pi))))\end{aligned} \quad \text{(B.2)}$$

- for each $B? \in \mathcal{A}_{FL(p)}$,

$$\mathcal{R}'(B?) = \{(| s |, | s |) \mid \mathcal{U}^F, s \models B\} \quad \text{(B.3)}$$

and otherwise $\mathcal{R}'(\alpha)$ is given inductively by the standard model conditions on α.

LEMMA B.8
$\mathcal{W}_{FL(p)}$ is finite and it has at most $2^{|FL(p)|}$ elements.

PROOF. For each $| s |$ we have the corresponding $FL(p)_s$. As a consequence: $\mathcal{W}_{FL(p)} \subseteq FL(p)$. Hence $| \mathcal{W}_{FL(p)} | \leq 2^{|FL(p)|}$.

∎

DEFINITION B.9 (Filtration)
$\mathcal{U}_{FL(p)}$ is called *FL(p)-Filtration* of $\mathcal{U}^F$ if the binary relation $\mathcal{R}(\alpha)'$ on $\mathcal{W}_{FL(p)}$ satisfies:

(F1) if $(s,t) \in \mathcal{R}(\alpha)$ then $(\mid s \mid, \mid t \mid) \in \mathcal{R}(\alpha)'$

(F2) if $(\mid s \mid, \mid t \mid) \in \mathcal{R}(\alpha)'$, then for all B
if $[\alpha]B \in FL(p)$ and $\mathcal{U}^F, s \models [\alpha]B$, then $\mathcal{U}^F, t \models B$

THEOREM B.10
$\mathcal{U}_{FL(p)}$ is a *FL(p)-Filtration* of $\mathcal{U}^F$

PROOF. We prove the theorem only for atomic actions, Σ, and Φ, since the proof for the other actions does not differ from the proof for PDL (see for example [23]).

- *Atomic Actions* which are not of the form $exec(\pi)$. Proof of the first filtration condition (F1). For each B such that $[a]B \in FL(p)$, for each $(s,t) \in \mathcal{R}(a)$ we have that $\mathcal{U}^F, s \models [a]B$ implies $\mathcal{U}^F, t \models B$ by definition of the semantics of $[a]B$. Moreover, notice that we can derive $[\alpha]\neg(Succ(\alpha) \wedge Fail(\alpha))$ by (NEC) and the axiom (NON-AMB). Hence, we can also prove that for each $(s,t) \in \mathcal{R}(a)$ we have $\mathcal{U}^F, t \models Ex(a) \wedge \neg(Succ(a) \wedge Fail(a))$ by the above formula and the theorem $[\alpha]Ex(\alpha)$ (i.e. (4.13)). As a consequence, when $(s,t) \in \mathcal{R}(a)$ we have that $\mathcal{U}^F, s \models [a]B$ implies $\mathcal{U}^F, t \models (B \wedge Ex(a) \wedge \neg(Succ(a) \wedge Fail(a)))$. But this is equivalent to $(\mid s \mid, \mid t \mid) \in \mathcal{R}(a)$ by the condition (B.1) of the definition of FL(p)-model. Hence we have proved the condition (F1).

  The proof of the second filtration condition (F2) easily follows from the condition (B.1) of the definition of FL(p)-model.

- $exec(\pi)$. Proof of the first filtration condition (F1). $(s,t) \in \mathcal{R}(exec(\pi))$ (i.e. $s \models \langle exec(\pi)\rangle True$) implies $s \not\models \pi = \bot$ thanks the axiom (EXEC-I). Furthermore for each B such that $[\alpha]B \in FL(p)$ and for each $\pi =$ "α" $\in FL(p)$ when we have $\mathcal{U}^F, s \models [\alpha]B$ and $\mathcal{U}^F, s \models \pi =$ "α" then $\mathcal{U}^F, s \models [exec(\pi)](B \wedge Ex(\alpha) \wedge \neg(Succ(\alpha) \wedge Fail(\alpha)))$. This is provable by the axiom (EXEC-II), the theorems $[\alpha]Ex(\alpha)$ (i.e. (4.13)) and $[\alpha]\neg(Succ(\alpha) \wedge Fail(\alpha))$ (obtained by (NEC) and (4.14)). Hence we have that for each $(s,t) \in \mathcal{R}(exec(\pi))$ we have that $\mathcal{U}^F, s \models [\alpha]B$ and $\mathcal{U}^F, s \models \pi =$ "α" imply $\mathcal{U}^F, t \models (B \wedge Ex(\alpha) \wedge \neg(Succ(\alpha) \wedge Fail(\alpha)))$. Taking into account that we can also prove that for each $(s,t) \in \mathcal{R}(exec(\pi))$, $\mathcal{U}^F, t \models (B \wedge Ex(exec(\pi)) \wedge \neg(Succ(exec(\pi)) \wedge Fail(exec(\pi))))$ as well as any other atomic action, we can apply the condition (B.2) and obtain $(\mid s \mid, \mid t \mid) \in \mathcal{R}'(exec(\pi))$ which proves the condition (F1).

  The condition (F2) easily follows from (B.2).

- Σ and Φ.
  - Let $(s,t) \in \mathcal{R}(\Sigma)$. Then if $B \in s$, so $[\Sigma]B \in s$ by axiom (NULL), hence $B \in t$. Thus $s \subseteq t$, and therefore $s = t$ as s is maximal. Thus we have $s = t$ implying $\mid s \mid = \mid t \mid$ that implies $(\mid s \mid, \mid t \mid) \in \mathcal{R}'(\Sigma)$ by definition of $\mathcal{R}'(\Sigma)$. Hence (F1) holds for Σ.
  - Let $(\mid s \mid, \mid t \mid) \in \mathcal{R}'(\Sigma)$. Then $\mid s \mid = \mid t \mid$. Thus if $[\Sigma]B \in FL(p)$ and $\mathcal{U}^F, s \models [\Sigma]B$, we have $\mathcal{U}^F, s \models B$, by axiom (NULL). Then $\mathcal{U}^F, t \models B$ since $\mid s \mid \in \rho_{FL(p)}(p)$ and $B \in FL(p)$. Hence (F2) holds for Σ.

  The proof for Φ is similar.

∎

LEMMA B.11 (Filtration Lemma)
For any $B \in FL(p)$

$$\mathcal{U}^F, s \models B \quad \text{iff} \quad \mathcal{U}_{FL(p)}, \mid s \mid \models B$$

PROOF. The Lemma is proved according to the usual proof given for modal and dynamic logics (see for example [23]). With respect to that proof we just have to add the proofs for the following step cases.

- $Succ(B?)$.

$\mathcal{U}^F, s \models Succ(B?)$	iff $\mathcal{U}^F, s \models B$	by axiom (SF-TEST)
	iff $\mathcal{U}_{FL(p)}, \mid s \mid \models B$	by inductive hypothesis
	iff $\mathcal{U}_{FL(p)}, \mid s \mid \models Succ(B?)$	by axiom (SF-TEST)

- $Fail(B?)$. The proof is similar to the proof for $Succ(B?)$.
- The proof for $Succ(\Sigma)$, $Fail(\Sigma)$, $Succ(\Phi)$, and $Fail(\Phi)$ are similar.

∎

COROLLARY B.12
$\mathcal{U}_{FL(p)}$ is a standard model.

PROOF. The definition of $\mathcal{U}_{FL(p)}$ guarantees all the conditions of standard models of $\mathcal{FSP}$ the following condition about tests:

$$\mathcal{R}'(B?) = \{(x, x) \mid \mathcal{U}_{FL(p)}, x \models B\}.$$

Nevertheless, the above condition holds for the Filtration lemma and the definition of $\mathcal{R}'(B?)$. ∎

THEOREM B.13 (Completeness)

For every $p \in \mathcal{P}$, if $\models p$ then $\vdash p$.

PROOF. Equivalently, we need to prove that if p is consistent, then it is satisfied in a standard model. In fact, the Lindenbaum's Lemma guarantees that if p is consistent then it is contained in a maximal set s, which is a state of the Canonical Model $\mathcal{U}^F$. By the Filtration Lemma, p is satisfied at state $\mid s \mid$ in the standard model $\mathcal{U}_{FL(p)}$. ∎

THEOREM B.14 (Decidability)

Validity in $\mathcal{FSP}$ is decidable.

PROOF. It is a consequence of the proof of Completeness Theorem and Lemma B.8. ∎

Received 23 September 1997