

A Logic for Acting, Sensing and Planning

Paolo Traverso
Mechanized Reasoning Group
IRST
38050 Povo, Trento
Italy

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

Abstract

We present a logic which allows us to reason about *acting*, and more specifically about *sensing*, i.e. actions that acquire information from the real world, and *planning*, i.e. actions that generate and execute plans of actions. This logic takes into account the fact that, as it happens in real systems, actions may fail, and provides the ability of reasoning about failure handling in acting, sensing and planning. We see this work as a first step towards a formal account of systems which are able to plan to act, plan to sense and plan to plan, and therefore, to integrate action, perception and reasoning.

1 Introduction

The idea of using logic for reasoning about actions and plans has been extensively studied in the past. So far, most of this research has mainly focused on two issues. The first is the problem of providing an adequate axiomatization of actions that a system can perform in the external environment. The second is the problem of providing a powerful and efficient deductive planning mechanism able to generate plans automatically. The idea underlying most of this research is that a logic can be used to predict action executability and effects, and therefore to generate "good" plans, i.e. plans that when executed are likely to achieve the desired goals.

Our research is related but different in focus. Our work starts from an analysis of planning systems that work in real world applications. Most of these systems, beyond plan generation, need to perform many different activities. For instance, they have to monitor executions, react to environmental changes, interleave planning, execution and perception, recover from failures, e.g. by replanning or by executing exception handling routines. In order to perform these activities, they need three basic capabilities: acting, sensing and planning. Acting capabilities are usually provided by a repertoire of actuators which perform actions in the external environment (e.g. wheels, gripper fingers and hands). Sensing capabilities are usually provided by a set of sensory devices which acquire information from the real world (e.g. sonars, odometers, cameras, microphones). Plan-

ning capabilities are usually provided by planning modules which generate plans to achieve given goals (e.g. modules for plan search, interactive systems for plan reuse) and execute plans. A planning system has to activate, coordinate and control all these devices and modules. Several systems which control acting, sensing and planning have been proposed so far (see for instance [Beetz and McDermott, 1994; Georgeff and Lansky, 1986; Simmons, 1990]) and have been successfully applied in particular application domains (like mobile robots and fault diagnosis for real time systems). In spite of this fact, no principled and theoretical account has been given of the behaviours of these systems.

The goal of this paper is to provide a logic which allows us to represent and reason about acting, sensing and planning. The motivation is twofold. First, the logic can be used to provide a specification of real systems which allows us to understand their requirements. Second, the logic can be used as a basic formal framework for building reasoning modules within real world applications. It can in fact be used to plan to act, plan to sense and plan to plan, and therefore to decide how to interleave acting, sensing and planning. We see this work as a first step towards a formal account of systems which are able to integrate reasoning, perception and action.

In order to achieve this goal, the logic we propose has some novel features. First, it is based upon an extended notion of *action*. The logic represents explicitly *sensing actions*, i.e. actions that acquire information and modify the state of knowledge of the agent, and *planning actions*, i.e. actions that generate and execute plans of actions. As a consequence, not only can the logic reason about the effects of actions in the real world, but also about the fact that a sensor has (not) been used to update the knowledge of the system about the world and the fact that the system has (not) a proper plan at hand which can be executed to try to achieve a given goal. Second, in real systems, 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. As a consequence, neither acting, nor sensing, nor planning is guaranteed to succeed. 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.

The paper is structured as follows. We describe the language of the logic in section 2 and its semantics in section 3. We give some axioms and theorems of the logic in section 4. We discuss some related work in section 5.

2 Language

The logic is a variation of process logic [Harel *et al.*, 1982], an extension of dynamic logic [Harel, 1984]. In section 2.1 we describe a language for action. In sections 2.2 and 2.3 we extend the language to represent sensing and planning actions, respectively.

2.1 Acting

The syntax of the logic is based upon two sets of symbols: \mathcal{P}_0 , the set of atomic (or basic) propositions and \mathcal{T}_0 , the set of atomic (or basic) tactics. From \mathcal{V}_0 and \mathcal{T}_0 we inductively construct the set \mathcal{V} of propositions and the set \mathcal{T} of tactics. \mathcal{V} and \mathcal{T} are the smallest sets such that:

1. $\mathcal{P}_0 \subseteq \mathcal{P}$; $True \in \mathcal{P}$;
2. If $p, q \in \mathcal{P}$, then $\neg p, p \wedge q \in \mathcal{P}$.
3. $W \in \mathcal{P}$;
4. If $p, q \in \mathcal{P}$ then $p \text{ chop } q \in \mathcal{P}$.
5. $\mathcal{T}_0 \subseteq \mathcal{T}$; $\Sigma, \Phi \in \mathcal{T}$;
6. If $\alpha, \beta, \gamma \in \mathcal{T}$, then **iffail** α then β else $\gamma \in \mathcal{T}$.
7. If $p \in \mathcal{P}$, $\alpha \in \mathcal{T}$, then $[\alpha]p \in \mathcal{P}$.
8. If $\alpha \in \mathcal{T}$, then $Fail(\alpha), Succ(\alpha) \in \mathcal{P}$.

We use \mathcal{V} , \rightarrow and \leftrightarrow as abbreviations in the standard way and, in addition, we abbreviate $\{a\}p$ to $\{a\}p$ as in dynamic logic¹.

Propositions are either true or false in *behaviours* (we say that they are behaviour propositions), where, intuitively, a behaviour is a finite sequence of states of the world. For example, if w_1, w_2, w_3, w_4 and w_5 are states, then $w_1 w_2 w_3 w_4 w_5$ is a behaviour. A single state is a particular case of behaviour, W is the proposition which holds over any behaviour which consists of a single state. The operator *chop* is applied to propositions p and q to yield a new proposition $p \text{ chop } q$. *chop* is used to reason about concatenations of behaviours, where, for example, the concatenation of $w_1 w_2 w_3$ and $W_3 W_4 w_5$ is $W_1 w_2 w_3 w_4 w_5$. $p \text{ chop } q$ holds in a behaviour b iff there exist two behaviours b_1 and b_2 such that the concatenation of b_1 and b_2 is b , p holds in b_1 and q holds in b_2 . We extend the language with the operator *last* which is applied to a proposition p to yield a new proposition $last(p)$.

$$last(p) \leftrightarrow True \text{ chop } (p \wedge W)$$

$last(p)$ holds in a behaviour b iff p holds in the final state of b .

Tactics represent actions. For instance, $goto(A)$ can be a basic (even if very complex) action which moves a

¹Actually, the language allows for tactics that include conditional expressions of the form *if p then a else B* and loops of the form *while p do a*. For lack of space, in this paper we do not describe the full syntax. The language, as well as its semantics, can be easily extended.

robot to a position A , e.g. by means of an in-door navigation system². Σ and Φ represent the primitive actions that generate success and failure, respectively. Σ (Φ) does nothing but terminate execution with success (failure). The intended meaning of **iffail** α then β else γ is: "do α , if α fails do β , otherwise do γ ". The intended meaning of $[\alpha]p$ is "every possible execution of α leads to a behaviour in which p holds". $Fail(\alpha)$ and $Succ(\alpha)$ hold iff α fails and succeeds, respectively. For any $\alpha \in \mathcal{T}$, we extend the language with the proposition $Ex(\alpha)$, defined as follows:

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

A proposition p can be seen as a goal to be achieved. Notice that success/failure of a tactic does not coincide necessarily with the achievement/not-achievement of a related goal. The former is a property of tactics, the latter is a relation between tactics and goals. A tactic may fail and, nevertheless, achieve the goal the tactic has been executed for. Vice versa, a tactic may succeed and may not achieve the goal. For example, let *put-on(a, b)* be a tactic that moves the block a on the block b . If a is on the block c , *put-on(a, b)* may fail, e.g. by moving a on the table, and nevertheless achieve the goal *Clear(c)*. If c is near b , *put-on(a, b)* may succeed and not achieve the goal *Far-from(a, c)*.

Constructs for failure handling are definable using Σ , Φ and **iffail**. For instance, we extend the language with the following definitions.

$$\begin{aligned} \alpha; \beta &= \text{iffail } \alpha \text{ then } \beta \text{ else } \beta, \\ \text{then}(\alpha, \beta) &= \text{iffail } \alpha \text{ then } \Phi \text{ else } \beta, \\ \text{orelse}(\alpha, \beta) &= \text{iffail } \alpha \text{ then } \beta \text{ else } \Sigma, \\ \text{repeat}(\alpha) &= \text{orelse}(\text{then}(\alpha, \text{repeat}(\alpha)), \Sigma) \end{aligned}$$

We define the usual construct $;$ for sequences of actions. In $a;/?$ the second action is executed anyway, independently of the failure/success of the first action. Notice that $;$ is a primitive construct in most of the logics proposed so far, e.g. in dynamic logic [Harel, 1984], in process logic [Harel *et al.*, 1982], in (extended versions of) situation calculus [Lesperance *et al.*, 1994] and in all theories of actions, e.g. in [Lifschitz, 1993]. This is due to the fact that $;$ constructs sequences without handling failure and these logics do not take into account failure. A sequential composition which takes into account failure is *then*. If the first action fails, *then* does not execute the second, but simply terminates execution with failure. *then* captures the behaviour of sequential executions in real systems where, if the first action fails, the second is not executed and control is passed to a module for failure recovery, or *else* is the construct for failure recovery. *orelse(a, B)* reacts to failure of a by executing B . *repeat* controls failure over the repeated execution of a tactic. It is recursively defined. It repeats the execution of a till a fails. If a never fails, execution does not terminate. Notice that if it terminates, *repeat(a)* always succeeds,

²More precisely, basic tactics are constructed from a set of terms, e.g. A , and a set of *tactic symbols*, that, intuitively represent action types, e.g. *goto*. For lack of space we skip the formal definition of the syntax of basic tactics.

since *orelse* in the definition of *repeat* reacts to failure by executing Σ , which always succeeds.

2.2 Sensing

We extend the language with a set C of symbols that we call *sensors*. For any sensor, we add a tactic and a proposition to the set of basic tactics and propositions.

If $c \in C$, then $sense(c) \in T_0$.

If $c \in C$, then $Sensed(c) \in P_0$.

Intuitively, sensors denote values which can be acquired through sensory devices. For instance, *wall-distance* can be the sensor which gets the value of the distance of the robot from the wall. We call $sense(c)$ a *sensing action* (for c). Its intended meaning is "acquire the value of the sensor c ". For example, $sense(wall-distance)$ can activate a sonar and/or a camera to acquire the value of *wall-distance*.

Sensing actions formalize activities that real systems have to perform extensively. Indeed, while most theories of actions are based on the assumption that, after acting is performed, the agent has at hand all the desired information about the new state of the world, this is not what happens in real systems. Most often, in real systems, the only way to get to know some facts about the state of affairs is to activate sensory devices and acquire information from the external environment, i.e. to execute sensing actions. Suppose for instance that a robot moves successfully to a given position A . At this point, the world around the robot has changed: its position has changed, as well as the distance from the wall, the distance from the nearest window, and so on. It is not realistic that the robot gets to know all these facts automatically after execution. For instance, after acting, knowledge about its position might be updated automatically, but it may have instead to measure (e.g. by a sonar or a camera) the distances from the wall, objects and landmarks to get to know their new values. Real systems have therefore to execute sensing actions explicitly. For instance, suppose that $goto(A)$ is a tactic which does not update the value of the sensor *wall-distance*. A possible tactic which moves the robot to A and then acquires the value of the distance from the wall is the following.

$goto(A); sense(wall-distance)$ (i)

A sensing action may fail. For instance, $sense(wall-distance)$ may fail since the sonar may not work or the camera may fail to detect the wall. In example (1), if sensing fails then the value of *wall-distance* is not updated.

Successful sensing actions update the state of knowledge of the agent since, at the end of their execution, the value of the sensor is acquired. This knowledge is expressed by the proposition $Sensed(c)$, which holds in any behaviour which is the final state of a successful sensing action for c . The idea here is that $Sensed(c)$ holds if we have just executed a successful sensing action for c and therefore the value of c is "up to date". For instance, consider the following propositions.

$Ex(goto(A)) \rightarrow \neg last(Sensed(wall-distance))$

$Ex(goto(B)) \rightarrow \neg last(Sensed(wall-distance))$ (3)

$Succ(goto(A); sense(wall-distance)) \rightarrow last(Sensed(wall-distance))$ (4)

$Ex(goto(A); sense(wall-distance); goto(B)) \rightarrow \neg last(Sensed(wall-distance))$ (5)

Proposition (2) states that $goto(A)$ does not update automatically the distance from the wall, i.e. after executing $goto(A)$ either with success or failure ($Ex(goto(A))$), then we get to a final state where *wall-distance* is not up to date ($last(Sensed(wall-distance))$). Proposition (3) is the analogous statement for the tactic $goto(B)$. The value of the sensor is up to date if the sensing action succeeds (proposition (4)). If we execute $goto(B)$ after sensing, then the value is not up to date (proposition (5)). Indeed, the last action might change the actual distance from the wall. Notice that the fact that the truth value of $Sensed(c)$ changes does not depend on the fact that the value of c changes or not. For instance, propositions (2)-(5) may hold even if *wall-distance* is constantly the same before and after the executions of $goto(A)$, $goto(B)$ and $sense(wall-distance)$.

2.3 Planning

We add to the language a set Π of symbols, that we call *names of tactics*, which is based upon an initial set of symbols π , and we extend the set of basic tactics and propositions as follows,

$\Pi_0 \subseteq \Pi$.

If $\pi \in \Pi_0$ and $p \in \mathcal{P}$, then $planfor(\pi, p) \in T_0$.

If $\pi \in \Pi$, then $exec(\pi) \in T_0$;

If $\pi \in \Pi_0$ and $p \in \mathcal{P}$, then $Planned(\pi, p) \in P_0$.

If $\alpha \in T$, then " α " $\in \Pi$.

We call " a " the *name* of the tactic a . Names of tactics denote tactics. For example, the name of $goto(A)$ denotes the syntactic expression $goto(A)$. The idea here is that planning generates a syntactic expression denoting a plan which can thereafter be executed. We call $planfor(\pi, p)$ a *plan generation action* (of π for p). Its intended meaning is: "generate a plan denoted by Π to achieve the goal p ". For example, if *Robot-at-A* is a proposition whose intended meaning is "the robot is in position A ", then $planfor(\pi, Robot-at-A)$ can be a tactic that generates the name n which denotes the simple plan $goto(A)$. We call $exec(Tr)$ a (*plan*) *execution action* (of IT). Its intended meaning is: "execute the plan denoted by Tr ". For example, the intended meaning of $exec\{goto(A)\}$ is: "execute the plan denoted by $goto(A)$ ". We call plan generation and execution actions, *planning actions*. As an example of combination of planning actions, consider the following tactic which generates a plan and then executes it.

$then(planfor(\pi_1, p), exec(\pi_1))$ (6)

Since plan execution actions may perform actions in the real world, they may fail. Failure in plan execution can be handled in different ways. Most classical planners (e.g. [Wilkins, 1985]) handle failure by replanning, i.e. by searching for a new plan. Reactive planners (e.g.

[Georgeff and Lansky, 1986; Simmons, 1990]) sometimes have no time for replanning, and therefore handle failure by executing precompiled special purpose exception handling routines. Our logic is expressive enough to represent these different failure handling mechanisms. For instance, tactic (7) reacts to failure of $exec(\pi)$ by replanning ($planfor(\pi_1, p)$), while tactic (8) reacts to failure by executing the plan denoted by π_2 , which can be a precompiled exception handling routine.

$$orelse(exec(\pi_1), planfor(\pi_1, p)) \quad (7)$$

$$orelse(exec(\pi_1), exec(\pi_2)) \quad (8)$$

In most classical planners, plan generation searches for a plan by using an internal model and does not operate in the real world. This is not what happens in most of the reactive planners (e.g. in [Georgeff and Lansky, 1986; Beetz and McDermott, 1994]) where, sometimes, the only way to decide for a plan is to do something in the world. For instance, a mobile robot which has to look for something in a building and does not know the map may have to "turn around the corner and open a door" in order to decide what to do next. In order to capture this extended notion of plan generation, $planfor$ has to be thought simply as a tactic which constructs a plan, with no constraints on whether it operates in the real world or not. As a consequence, in our view, plan generation may fail in the same way as acting, sensing and plan execution may fail. The constructs for failure handling defined in section 2.1 can be used to handle failure in plan generation. For instance, in example (6), if $planfor$ fails, $then$ captures failure and does not execute TTJ.

Successful plan generation actions update the state of knowledge of the agent since they produce a plan that is available for execution. This knowledge is expressed by the proposition $Planned(\pi, p)$, which holds in any behaviour which is the final state of a successful plan generation action of π for p . The idea here is that $Planned(\pi, p)$ holds if we have just executed a plan generation action of Π for p and therefore the plan at hand, i.e. the plan denoted by π , is "the proper plan for that situation". For instance, consider the following propositions.

$$Ex(goto(A)) \rightarrow \neg last(Planned(\pi_1, p)) \quad (9)$$

$$Succ(planfor(\pi_1, p)) \rightarrow last(Planned(\pi_1, p)) \quad (10)$$

$$Ex(planfor(\pi_1, p); goto(A)) \rightarrow \neg last(Planned(\pi_1, p)) \quad (11)$$

$$Succ(planfor(\pi_1, p); exec(\pi_1); planfor(\pi_1, p)) \rightarrow last(Planned(\pi_1, p)) \quad (12)$$

Proposition (9) states that moving the robot to A makes the plan denoted by Π obsolete. Proposition (10) states that success in planning leads to a final state where the plan at hand is "the proper plan for that situation". Proposition (11) states that after planning and executing $goto(A)$, the plan denoted by Π_1 may be obsolete (even if still available for execution). Indeed, acting in the world may change the world and invalidate old plans. Proposition (12) states that after plan generation, plan execution, and finally plan generation again, the plan at hand is "the plan for that situation".

Two remarks are in order. First, having at hand "the plan for that situation" does not mean that the execution of the plan will actually 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 has achieved the goal or not. Second, the second occurrence of $planfor(\pi_1, p)$ in (12) might or might not generate a new tactic different from the one generated by the first occurrence of $planfor(\pi_1, p)$, i.e. the tactic denoted by π_1 might or might not change. This depends on how $planfor$ behaves and, for instance, on whether the execution of π_1 changes the world in a way that influences the second plan generation action.

3 Semantics

The semantics of the language is defined relative to a given structure \mathcal{U} , of the form

$$\mathcal{U} = \langle \mathcal{D}, \mathcal{W}, \mathcal{I} \rangle$$

where \mathcal{D} is the domain of interpretation, \mathcal{W} is an abstract set of states and \mathcal{I} is the interpretation function. $\mathcal{D} = \mathcal{D}_o \cup \mathcal{T}_u$. \mathcal{D}_o is used to interpret sensors and \mathcal{T}_u is used to interpret names of tactics. The set of behaviours \mathcal{B} is the set of all finite-length sequences of states, repetitions allowed, i.e. $\mathcal{B} = \mathcal{W}^+$. Behaviour concatenation is defined as follows: if $w_i \in \mathcal{W}$, $i = 1, \dots, j-1, j, j+1, \dots, n$, $b_1, b_2 \in \mathcal{B}$, $b_1 = w_1 \dots w_j$, $b_2 = w_j \dots w_n$, then $b_1 \cdot b_2 = w_1 \dots w_{j-1} w_j w_{j+1} \dots w_n$. Concatenation is extended over sets of behaviours in the usual way.

\mathcal{I} assigns interpretations to sensors and names depending on behaviours. For any $b \in \mathcal{B}$, we write c_b and π_b the interpretation of $c \in \mathcal{C}$ and $\pi \in \Pi$ in the behaviour b , respectively. For any $c \in \mathcal{C}$, $c_b \in \mathcal{D}_o$, i.e. in any behaviour, a sensor denotes an element of \mathcal{D}_o , that we call *the value of the sensor*. For any $\pi \in \Pi$, $\pi_b \in \mathcal{T}_u$, i.e. a name of a tactic denotes an element of \mathcal{T}_u , where \mathcal{T}_u is the set of tactics in \mathcal{U} . Notice that we have the set of tactics \mathcal{T} in the language and the set of tactics \mathcal{T}_u in the structure \mathcal{U} . \mathcal{T} and \mathcal{T}_u contain exactly the same elements. We need \mathcal{T}_u in order to interpret names of tactics. For any behaviour b , " α " $_b = \alpha$.

\mathcal{I} assigns subsets of \mathcal{B} to propositions and tactics. We write $\rho(p)$ and $\mathcal{R}(\alpha)$ the sets of behaviours assigned by \mathcal{I} to $p \in \mathcal{P}$ and $\alpha \in \mathcal{T}$. We write $b \models p$ iff $b \in \rho(p)$. $\mathcal{R}(\alpha)$ is divided into two subsets:

$$\mathcal{R}(\alpha) = \mathcal{S}(\alpha) \cup \mathcal{F}(\alpha) \quad (13)$$

$\mathcal{S}(\alpha)$ is the set of successful behaviours, that we call the *success set* of α , and $\mathcal{F}(\alpha)$ is the set of behaviours that fail, that we call the *failure set* of α . \mathcal{I} assigns an arbitrary subset of \mathcal{B} to each basic proposition p and arbitrary disjoint success and failure sets to basic tactics, i.e.

$$\forall \alpha \in \mathcal{T}_0 \quad \mathcal{S}(\alpha) \cap \mathcal{F}(\alpha) = \emptyset \quad (14)$$

We have the following condition over $\mathcal{R}(\alpha)$ of basic tactics:

$$\forall \alpha \in \mathcal{T}_0 \quad \forall b_1, b_2 \in \mathcal{R}(\alpha) \quad \forall b_3 \in \mathcal{B} \quad (15)$$

$$\text{if } b_3 \notin \mathcal{W}, \text{ then } b_1 \neq b_2 \cdot b_3$$

This condition states that we cannot have a behaviour of an action whose final state is the same as the intermediate state of a behaviour of the same action. This corresponds to the fact that the agent, in a given state of its computation machinery, always stops or always continues execution. Conditions (14) and (15) allow us to prove that success and failure sets are mutually exclusive:

$$\forall \alpha \in \mathcal{T} \quad \mathcal{S}(\alpha) \cap \mathcal{F}(\alpha) = \emptyset \quad (16)$$

and that condition (15) holds for any tactic.

\mathcal{I} is extended inductively to supply meanings for the full sets \mathcal{P} and \mathcal{T} .

1. $b \models \text{True}$;
2. $b \models \neg p$ iff $b \not\models p$; $b \models p \wedge q$ iff $b \models p$ and $b \models q$
3. $b \models W$ iff $b \in \mathcal{W}$;
4. $b \models p \text{ chop } q$ iff $\exists b_1, b_2 \in \mathcal{B}$ such that $b = b_1 \cdot b_2$ and $b_1 \models p$ and $b_2 \models q$.
6. $\mathcal{S}(\Sigma) = \mathcal{W}$; $\mathcal{F}(\Sigma) = \emptyset$; $\mathcal{S}(\Phi) = \emptyset$; $\mathcal{F}(\Phi) = \mathcal{W}$.
7. $\mathcal{S}(\text{iffail } \alpha \text{ then } \beta \text{ else } \gamma) = \mathcal{S}(\alpha) \cdot \mathcal{S}(\gamma) \cup \mathcal{F}(\alpha) \cdot \mathcal{S}(\beta)$;
 $\mathcal{F}(\text{iffail } \alpha \text{ then } \beta \text{ else } \gamma) = \mathcal{S}(\alpha) \cdot \mathcal{F}(\gamma) \cup \mathcal{F}(\alpha) \cdot \mathcal{F}(\beta)$.
8. $b \models [\alpha]p$ iff $\forall b' \in \mathcal{R}(\alpha)$, then $b \cdot b' \models p$.
9. $b \models \text{Fail}(\alpha)$ iff $b \in \mathcal{F}(\alpha)$;
 $b \models \text{Succ}(\alpha)$ iff $b \in \mathcal{S}(\alpha)$;

Some remarks. W is interpreted into \mathcal{W} . Notice that W is the identity *w.r.t.* concatenation, i.e. if B is any set of behaviours, then $B \cdot W = W \cdot B = B$. The failure set of Σ is empty. This corresponds to the fact that it always succeeds. Its success set is \mathcal{W} . This corresponds to the fact that Σ does nothing else but succeeding. Vice versa, the success set of Φ is empty while its failure set is \mathcal{W} .

The constructs *;*, *then*, *orelse* and *repeat* inherit their meanings from their definitions in terms of Σ , Φ and *iffail*. For lack of space we do not describe the semantics of *repeat*, which is recursively defined and interpreted by building a fix point. We give below the success and failure sets for *;*, *then* and *orelse*.

$$\begin{aligned} \mathcal{S}(\alpha; \beta) &= \mathcal{R}(\alpha) \cdot \mathcal{S}(\beta) \\ \mathcal{F}(\alpha; \beta) &= \mathcal{R}(\alpha) \cdot \mathcal{F}(\beta) \\ \mathcal{S}(\text{then}(\alpha, \beta)) &= \mathcal{S}(\alpha) \cdot \mathcal{S}(\beta) \\ \mathcal{F}(\text{then}(\alpha, \beta)) &= \mathcal{F}(\alpha) \cup \mathcal{S}(\alpha) \cdot \mathcal{F}(\beta) \\ \mathcal{S}(\text{orelse}(\alpha, \beta)) &= \mathcal{S}(\alpha) \cup \mathcal{F}(\alpha) \cdot \mathcal{S}(\beta) \\ \mathcal{F}(\text{orelse}(\alpha, \beta)) &= \mathcal{F}(\alpha) \cdot \mathcal{F}(\beta) \end{aligned}$$

Notice that, in case both α and β succeed, *then*(α, β) behaves like $\alpha; \beta$. In case of failure of the first action, the failure set of *then*(α, β) is the failure set of the first action, i.e. $\mathcal{F}(\alpha)$. This captures the idea that when the first action fails, the second is not executed. Analogously, notice that when the first action succeeds, $\mathcal{S}(\text{orelse}(\alpha, \beta))$ is the success set of α , i.e. $\mathcal{S}(\alpha)$, since no recovery from failure is needed. When α fails, β is executed and *orelse*(α, β) might either fail or succeed, depending on the failure/success of β .

In the following, we give interpretation to *Sensed*(c) and *Planned*(π, p).

$$\begin{aligned} b \models \text{Sensed}(c) & \quad \text{iff} \quad \exists b_1 \in \mathcal{S}(\text{sense}(c)) \\ & \quad \text{such that } b_1 \cdot b = b_1. \\ b \models \text{Planned}(\pi, p) & \quad \text{iff} \quad \exists b_1 \in \mathcal{S}(\text{planfor}(\pi, p)) \\ & \quad \text{such that } b_1 \cdot b = b_1. \end{aligned}$$

Sensed(c) holds in all and only the states which are final states of successful behaviours of *sense*(c). *Planned*(π, p) holds in all and only the states which are final states of successful behaviours of *planfor*(π, p).

Notice that, in order to keep the semantics general, we have not given any condition on the interpretations of sensing and planning actions. They are basic tactics, and as such they are assigned arbitrary disjoint success and failure sets. Their semantics depends on the particular sensing, planning and execution mechanisms which are available to the system. Nevertheless, there are conditions on planning that seem reasonable. We give some of these conditions below.

- I. $\mathcal{S}(\text{exec}(\alpha)) = \mathcal{S}(\alpha)$; $\mathcal{F}(\text{exec}(\alpha)) = \mathcal{F}(\alpha)$.
- II. $\mathcal{S}(\text{planfor}(\pi, p)) = \{b \mid \exists b' \in \mathcal{R}(\pi_b) \text{ such that } b \cdot b' \models p\}$

Condition I states that execution, given a name denoting a tactic, does nothing but executing the tactic itself. Condition II states that there must exist at least one behaviour b' in the execution of a tactic denoted by π (i.e. π_b) and generated by the planning action *planfor*(π, p), such that behaviours in the successful set of *planfor*(π, p) concatenated with b' achieve the goal p we have planned for (i.e. $b \cdot b' \models p$). This condition allows only for plan generations that produce a plan that, when executed, has at least a possibility of achieving the desired goal.

4 Some theorems

Given the language and its semantics, the goal is to provide a deductive system which is complete and correct. In the following, we give some axiom schemas which we have proved correct *w.r.t.* the given semantics and which allow us to prove some interesting theorems about acting, sensing and planning. We start with axioms inherited from dynamic and process logic:

$$\begin{aligned} (A1) \quad & [\alpha](p \wedge q) \leftrightarrow ([\alpha]p \wedge [\alpha]q) \\ (A2) \quad & [\alpha](p \rightarrow q) \leftrightarrow ([\alpha]p \rightarrow [\alpha]q) \end{aligned}$$

Some basic properties of W and *chop*:

$$\begin{aligned} (A3) \quad & ((p \text{ chop } W) \leftrightarrow p) \wedge ((W \text{ chop } p) \leftrightarrow p) \\ (A4) \quad & ((p \text{ chop } q) \text{ chop } r) \leftrightarrow (p \text{ chop } (q \text{ chop } r)) \end{aligned}$$

Some axioms to reason about failure and success:

$$\begin{aligned} (A5) \quad & W \rightarrow [\alpha]Ex(\alpha). \\ (A6) \quad & \neg(\text{Succ}(\alpha) \wedge \text{Fail}(\alpha)). \\ (A7) \quad & Ex(\alpha) \rightarrow \neg(Ex(\alpha) \text{ chop } \neg W) \\ (A8) \quad & (\neg \text{Fail}(\Sigma)) \wedge (\text{Succ}(\Sigma) \leftrightarrow W) \\ (A9) \quad & (\neg \text{Succ}(\Phi)) \wedge (\text{Fail}(\Phi) \leftrightarrow W) \\ (A10) \quad & ([\Sigma]p \leftrightarrow p) \wedge ([\Phi]p \leftrightarrow p) \\ (A11) \quad & \text{Succ}(\text{iffail } \alpha \text{ then } \beta \text{ else } \gamma) \leftrightarrow \\ & ((\text{Fail}(\alpha) \text{ chop } \text{Succ}(\beta)) \vee \\ & (\text{Succ}(\alpha) \text{ chop } \text{Succ}(\gamma))) \end{aligned}$$

$$(A12) \text{Fail}(\text{iffail } \alpha \text{ then } \beta \text{ else } \gamma) \leftrightarrow \\ ((\text{Fail}(\alpha) \text{ chop } \text{Fail}(\beta)) \vee \\ (\text{Succ}(\alpha) \text{ chop } \text{Fail}(\gamma))) \\ (A13) W \rightarrow ([\text{iffail } \alpha \text{ then } \beta \text{ else } \gamma]p \leftrightarrow \\ ([\alpha]\text{Fail}(\alpha) \rightarrow [\alpha](\beta)p) \wedge \\ ([\alpha]\text{Succ}(\alpha) \rightarrow [\alpha](\gamma)p))$$

(A5) states that, if we are in the initial state of a behaviour of α , then the execution of α either leads to success or failure. (A6) states that success and failure are mutually exclusive. It corresponds to the condition $\mathcal{S}(\alpha) \cap \mathcal{F}(\alpha) = \emptyset$. (A7) states condition (15) for any tactic. Indeed, for any $p \in \mathcal{P}$, for any $b = w_1 \dots w_n$, $b \models p \text{ chop } \neg W$ iff $w_1 \dots w_k \models p$ with $1 \leq k \leq n-1$. (A7) states therefore that, if $w_1 \dots w_n \in \mathcal{R}(\alpha)$ then $w_1 \dots w_k \notin \mathcal{R}(\alpha)$, with $1 \leq k \leq n-1$. (A8) and (A9) state that Σ (Φ) never fails (succeeds) and that the success (failure) set of Σ (Φ) is \mathcal{W} . (A10) states that Σ and Φ do not change the world, i.e. the set of properties which hold before the execution of Σ and Φ also hold after their execution and vice versa. (A11) and (A12) describe the success and failure set of *iffail*. (A13) states that p holds after *iffail* α then β else γ iff it holds after α and β if α fails, and after α and γ if α succeeds.

The following axioms describe sensing and planning.

$$(A14) \text{Succ}(\text{sense}(c)) \leftrightarrow \\ (\text{Succ}(\text{sense}(c)) \text{ chop } \text{Sensed}(c)) \\ (A15) \text{Succ}(\text{planfor}(\pi, p)) \leftrightarrow \\ (\text{Succ}(\text{planfor}(\pi, p)) \text{ chop } \text{Planned}(\pi, p))$$

(A14) and (A15) state that sensing and planning with success end up in a final state where *Sensed*(c) and *Planned*(π, p) hold, respectively.

Conditions I and II (section 3) are stated as follows.

$$(A16) ([\text{exec}(\alpha)]p \leftrightarrow [\alpha]p) \wedge \\ (\text{Succ}(\text{exec}(\alpha)) \leftrightarrow \text{Succ}(\alpha)) \wedge \\ (\text{Fail}(\text{exec}(\alpha)) \leftrightarrow \text{Fail}(\alpha)) \\ (A17) \text{Succ}(\text{planfor}(\pi, p)) \rightarrow (\text{exec}(\pi))p$$

(A16) states that the effects of *exec*(α) are the same as the effects of α . (A17) states that plan generation generates a plan that has at least one possibility of achieving the desired goal.

From (A1)-(A13), we can prove the following theorems which can be used to reason about success and failure of the constructs *;*, *then* and *orelse*.

$$(T1) \text{Succ}(\alpha; \beta) \leftrightarrow ((\text{Fail}(\alpha) \text{ chop } \text{Succ}(\beta)) \vee \\ (\text{Succ}(\alpha) \text{ chop } \text{Succ}(\beta))) \\ (T2) \text{Fail}(\alpha; \beta) \leftrightarrow ((\text{Fail}(\alpha) \text{ chop } \text{Fail}(\beta)) \vee \\ (\text{Succ}(\alpha) \text{ chop } \text{Fail}(\beta))) \\ (T3) \text{Succ}(\text{then}(\alpha, \beta)) \leftrightarrow \\ (\text{Succ}(\alpha) \text{ chop } \text{Succ}(\beta)) \\ (T4) \text{Fail}(\text{then}(\alpha, \beta)) \leftrightarrow \\ (\text{Fail}(\alpha) \vee (\text{Succ}(\alpha) \text{ chop } \text{Fail}(\beta))) \\ (T5) \text{Succ}(\text{orelse}(\alpha, \beta)) \leftrightarrow \\ (\text{Succ}(\alpha) \vee (\text{Fail}(\alpha) \text{ chop } \text{Succ}(\beta))) \\ (T6) \text{Fail}(\text{orelse}(\alpha, \beta)) \leftrightarrow \\ (\text{Fail}(\alpha) \text{ chop } \text{Fail}(\beta))$$

From (A1)-(A14), we prove the following theorems.

$$(T7) \text{Succ}(\text{sense}(c)) \rightarrow \text{last}(\text{Sensed}(c))$$

$$(T8) \text{Succ}(\alpha; \text{sense}(c)) \rightarrow \text{last}(\text{Sensed}(c))$$

Moreover, from (T8) we can prove (4) and, given (3) as an axiom, we can prove (5) from (T1) and (T2). From (A1)-(A13) and (A15), we prove the following theorems.

$$(T9) \text{Succ}(\text{planfor}(\pi, p)) \rightarrow \text{last}(\text{Planned}(\pi, p)) \\ (T10) \text{Succ}(\alpha; \text{planfor}(\pi, p)) \rightarrow \text{last}(\text{Planned}(\pi, p))$$

which can be used to prove propositions (10) and (12). (9), (T1) and (T2) can be used to prove proposition (11).

5 Related work

This paper is an elaboration and extension of the intuitions originally presented in [Giunchiglia *et al.*, 1994].

Compared to the previous research in theories of actions, the work described in this paper is limited in at least three respects. First, we do not allow for variables and quantifiers in our logic. Second, we do not deal with asynchronous and parallel events and actions. Third, we do not discuss how our logic deals with the frame problem. Major future goals include these issues. However, these issues, though very important, are somehow orthogonal to the main message of this paper, which is about describing a theory of acting, sensing, and planning, i.e. a theory which integrates in a uniform framework important basic features of planning systems for real world applications. As far as we know, the approach presented in this paper has never been proposed before.

The closest work on failure is that described in [Rao and Georgeff, 1991]. [Rao and Georgeff, 1991] presents a formal framework for BDI-architectures and commitment (we do not deal with these issues in this paper) which represents explicitly failure and success of events. *succeeds*(e), *fails*(e), *succeeded*(e) and *failed*(e) are state formulas (of a propositional branching time logic) which express immediate future and past performance, respectively successfully and unsuccessfully, of event e . Semantically, arc functions S_w and F_w map adjacent time points to the event that occurred with success or with failure. Technically, our approach is different since *Succ*(a) and *Fail*(a) are behaviour propositions. This captures the fact that the execution of an action may result in different sequences of states, and the fact that its failure and success may depend on the whole sequence and not only on a single state. The main conceptual difference is in the focus and objectives of the two works. We are interested in a framework for failure handling, i.e. in how actions can be composed through constructs which capture and react to failure, since we see flexible failure recovery as one of the major activities that real planning systems have to perform. For this reason we have a logic which combines actions through $\langle \mathcal{E}, E, \text{ifTail} \rangle$, *;*, *then*, *or else* and *repeat*. While the theory proposed in [Rao and Georgeff, 1991] does not deal with action composition, and more important, with constructs for failure handling. Moreover, our logic allows us to express and reason about sensing and planning actions, while [Rao and Georgeff, 1991] does not.

In [Lesperance *et al.*, 1994], situation calculus is extended with complex actions, e.g. sequences, conditionals and loops, and with "perception actions", or "knowledge producing actions", of the form *SENSE* p and *READ*,-

where P and r are a fluent and a term, respectively. Knowledge about perception is expressed by means of wffs of the form $Knows(P,a)$ and $Kref(r, s)$, where $*$ is a situation. Technically, our approach is different since we have no situations in the logic. Moreover, in this paper we have described a class of sensing actions which is less expressive than the class of perception actions defined in [Lesperance *et al.*, 1994]. However, the given language, semantics and axiomatization can be easily extended to include sensing actions about propositions and terms. Conceptually, our work differs mainly in two aspects. First, our logic captures the fact that perception is a complex task that, when it has to be performed by real systems, is not guaranteed to succeed. Actions of the form $sense(c)$ may actually fail to acquire information and therefore fail to produce knowledge. As a consequence, in our logic we do not have theorems analogous to $Kref(r,do(READ,r,s))$, which can be read as "after doing READ, the agent knows the denotation of r ", but we can prove theorem (T7), i.e. "if sensing succeeds, then the agent has sensed c ". Second, our logic allows us to express and reason about planning actions, while [Lesperance *et al.*, 1994] does not.

The closest work on planning actions is that described in [Steel, 1994b] (see also [Steel, 1994a]), where dynamic and epistemic logic are used to express formulas of the form "plan to do an action that achieves a goal, then do it". The main differences with our work are the following. First, in [Steel, 1994b] plan generation is seen as "specialization" of "non operational" actions, i.e. actions which cannot be executed. We see instead plan generation as an executable action which constructs a plan. This corresponds to the fact that systems have planning modules which can be activated to generate plans. Second, in [Steel, 1994b], planning is seen as an action which does not affect the external environment while we do not rely on this assumption since many reactive planners have to generate plans by acting in the world. Finally, our logic deals explicitly with failure handling in plan generations and executions.

The logic we have proposed is based on work on MRG [Giunchiglia *et al.*, 1991; Traverso *et al.*, 1992], a reactive planning system which executes tactics. At the moment, MRG is used in a large scale, real world application under development at IRST. This application aims at the development of a system that has to control and coordinate mobile robots, navigating in unpredictable environments inhabited by humans and performing high level tasks, like transportation tasks in hospitals and offices. MRG tactics are executed by means of systems that perform sensing and acting, e.g. a reactive sensor and actuator controller for navigation tasks and a system for speech recognition, and by means of systems that generate and execute plans, e.g. path planners and activity schedulers. We plan to use the formal framework to extend the functionalities of MRG and to specify the requirements of the application under development.

Acknowledgments

We thank Fausto Giunchiglia for his invaluable support, and for all his many conceptual and technical comments

on early drafts of this paper. They have contributed to improve the paper significantly. We also thank Luciano Serafini for fruitful discussions.

References

- [Beetz and McDermott, 1994] M. Beetz and D. McDermott. Improving Robot Plans During Their Execution. In *Proceedings 2nd International Conference on AI Planning Systems (AIPS-94)*, Chicago, 1994.
- [Georgeff and Lansky, 1986] M. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings IEEE*, 74(10):1383-1398, 1986.
- [Giunchiglia *et al.*, 1991] F. Giunchiglia, P. Traverso, A. Cimatti, and L. Spalazzi. Programming planners with flexible architectures. Technical Report 9112-19, IRST, Trento, Italy, 1991.
- [Giunchiglia *et al.*, 1994] F. Giunchiglia, L. Spalazzi, and P. Traverso. Planning with Failure. In *Proceedings 2nd International Conference on AI Planning Systems (AIPS-94)*, Chicago, 1994.
- [Hard *et al.*, 1982] D. Harel, D. Kozen, and R. Parikh. Process Logic: expressiveness, decidability, completeness. *Journal of computer and system sciences*, 25:144-170, 1982.
- [Harel, 1984] D. Harel. Dynamic Logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume II, pages 497-604. D. Reidel Publishing Company, 1984.
- [Lesperance *et al.*, 1994] Y. Lesperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R.B. Scherl. A Logical Approach to High-Level Robot Programming- A Progress Report. In *Control of the physical world by intelligent systems, working notes of the AAAI Fall Symp.*, 1994.
- [Lifschitz, 1993] V. Lifschitz. A Language for Describing Actions. In *Proceedings 2nd Symposium on Logical Formalizations of Commonsense Reasoning*, Austin, 1993.
- [Rao and Georgeff, 1991] A. S. Rao and M. P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In *Proceedings KR'91, Principle of Knowledge Representation and Reasoning*, pages 473-484, Cambridge Massachusetts, 1991. Morgan Kaufmann.
- [Simmons, 1990] 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.
- [Steel, 1994a] S. Steel. Action under Uncertainty. *Journal of Logic and Computation, Special Issue on Action and Processes*, 4(5):777-795, 1994.
- [Steel, 1994b] S. Steel. Planning to Plan, 1994. Technical Report, Dept Computer Science, University of Essex, Colchester C04 3SQ, UK.
- [Traverso *et al.*, 1992] P. Traverso, A. Cimatti, and L. Spalazzi. Beyond the single planning paradigm: introspective planning. In *Proceedings ECAI-92*, pages 643-647, Vienna, Austria, 1992. IRST-Technical Report 9204-05, IRST, Trento, Italy.
- [Wilkins, 1985] D.E. Wilkins. Recovering from execution errors in SIPE. *Computational Intelligence*, 1:33-45, 1985.