A Multivalued Logic Approach to Integrating Planning and Control

Alessandro Saffiotti,[1] Kurt Konolige and Enrique H. Ruspini

Artificial Intelligence Center
SRI International
Menlo Park, CA 94025, U.S.A.
{*saffiotti,konolige,ruspini*}*@ai.sri.com*

Abstract

Intelligent agents embedded in a dynamic, uncertain environment should incorporate capabilities for both planned and reactive behavior. Many current solutions to this dual need focus on one aspect, and treat the other one as secondary. We propose an approach for integrating planning and control based on *behavior schemas*, which link physical movements to abstract action descriptions. Behavior schemas describe behaviors of an agent, expressed as trajectories of control actions in an environment, and goals can be defined as predicates on these trajectories. Goals and behaviors can be combined to produce conjoint goals and complex controls. The ability of multivalued logics to represent graded preferences allows us to formulate tradeoffs in the combination. Two composition theorems relate complex controls to complex goals, and provide the key to using standard knowledge-based deliberation techniques to generate complex controllers. We report experiments in planning and execution on a mobile robot platform, Flakey.

[1] Current address: IRIDIA, Université Libre de Bruxelles, 50 av. Roosevelt CP 194/6, B-1050 Brussels, Belgium. E-mail: asaffio@ulb.ac.be

Artificial Intelligence 76 (1995) — in press.

1 Introduction

Mobile robots are becoming increasingly sophisticated and autonomous, and we demand more intelligent behavior from them in complex environments. To meet these expectations, we must address a set of problems associated with real-world environments: knowledge of the environment is partial and approximate; sensing is partial and noisy; the dynamics of the environment can be only partially predicted; and an agent's hardware execution is not completely reliable. Classical AI planning approaches to these problems are inadequate, especially in providing realtime decision-making and control for the robot. On the other hand, purely reactive systems of control (e.g., [Bro82,Fir87,KR90]), while providing immediate response to unpredicted environmental situations, cannot totally substitute for planned behavior in solving complex tasks (for example, by deciding not to carry an oil lantern downstairs to look for a gas leak[Fir87].)

One solution to the dual need for planning and reactivity is to adopt a two-level model: at the upper level, a planner decides a sequence of abstract goals to be achieved, based on the available knowledge; at the lower level, a complex controller achieves these goals while dealing with the environmental contingencies (e.g.,[McD90,Con92,Gat92,Ark90,PRK90]). The controller is "complex" because it must be able to simultaneously satisfy strategic goals coming from the planner (e.g., going to the end of the corridor), and low-level "innate" goals (e.g., avoiding obstacles on the way). It is the controller's job to produce physical movements that satisfy these goals to the highest degree possible. Thus, the two main challenges to developing a complex controller are trading off between multiple goals, and bridging the gap between abstractly specified goals and physical movements.

A reasonable approach to complex control is to decompose the problem into small units of control, each of which implements one specific motor skill, and results in a certain type of physical movement [Ark87,Bro82]. In the psychological literature, the concept of coordinated activity directed at a particular goal has been given the name *schema*. Arbib and his colleagues have exploited the schema concept to model brain functioning and robot control. For example, Arbib and House [AH85] describe how schemas can account for the prey-seeking and obstacle-avoidance behavior of frogs. Each of these two task-level goals is represented as a potential field attracting or repelling the frog; the combination of the fields describes the movements of the frog towards the prey and away from obstacles. Further extensions of this idea to robot control were investigated by Lyons and Arbib [LA85], Overton [Ove84], and Arkin [Ark90] under the name *motor schema*.

Arbib's motor schemas give an answer to trading off among multiple goals,

by providing a modular decomposition of complex control problems. They do not, however, try to answer the second challenge above, of bridging the gap between symbolically specified goals and low-level physical movements. In this paper, we develop a compositional approach to behavior, based on the mathematics of multivalued logic, that directly connects to higher-level planning and deliberation processes. The starting point of our approach are very close to those of Arbib's. In fact, one can view our approach as a mathematically-motivated reconstruction of motor schemas, using the tools of multivalued logic. The advantages to this formalization are the following.

- Behaviors are described in terms of preferences among control actions. Hence, multiple behaviors can be composed mathematically as a means of trading off between their goals — a more powerful form of coordination than the linear superposition used in motor schemas.
- The concept of a *goal* as a predicate on the trajectory produced by the controller arises naturally, and bridges the gap between physical movements and more abstract descriptions of action.
- Complex goals can be achieved by composing their respective behaviors. Thus, standard deliberation procedures, such as goal-regression planning, can be used to generate complex controllers.

1.1 Outline of the approach

We develop the theory of behavior in a bottom-up fashion, describing movements at the following levels of abstraction:

bodily movement	*straighten elbow joint*	$\Rightarrow$
execution in a specific environment	*put arm out car window*	$\Rightarrow$
goal satisfaction	*signal left turn*	

This way of connecting movements, actions and achievements has been inspired by the work of Israel, Perry and Tutiya [IPT91]. Formally, we use the framework of multivalued logics [LT83,Res69], which allows us to express partial preferences and to combine them by using logical operators [BZ70,Rus91a].

The basic units of control, *control schemas*, are descriptions of types of movements. Control schemas define the agent's basic movement capabilities, like making a step or aiming the camera to the right. Like classical controllers, control schemas relate internal states to effector commands. However, control schemas are less committal than classical controllers in that they map each state to a measure of preference, or *desirability function*, over the space of all possible commands. The idea here is that different commands can generate, to

a greater or lesser extent, the same type of movement. Desirability functions are represented by predicates in a multivalued logic.

We compose control schemas by combining the corresponding desirability functions via the operators of multivalued logics. For instance, we can compose control schemas to obtain a movement that will, under some conditions, satisfy the desirability functions of both behaviors. Care must be taken, however, when the control schemas are conflicting, that is, they have no common preferences. For example, if a robot is facing an obstacle blocking a corridor, an obstacle avoidance control schema may want to stop, while a corridor following schema may want to go forward. In general, we associate each control schema with its own *context* of applicability: e.g., when the corridor is blocked, plain corridor following cannot be used, and its preferences should be disregarded. We express contextual conditions by using formulae of a multivalued logic; and use the degrees of truth of these formulae to weight the preferences of different schemas. This weighted combination, called *context-depending blending,* is the main coordination mechanism for control schemas.

Control schemas operate with respect to the agent's internal state, including its sensor readings. Planners, on the other hand, manipulate action descriptions that contain representations of objects in the world, e.g., picking up block "A." To reconcile these two levels of abstraction, we insert some parts of the planner's model into the controller, e.g., the expected size and position of the intended block. These object models, or *descriptors*, are then used as input to control schemas, "lifting" the effects of the controller to the level of abstraction used by the planner. To maintain a closed-loop response, we keep track of these objects during execution, by polling perceptual routines to keep the descriptors *anchored* to their real-world correspondent. A control schema, together with a set of object descriptors and a contextual condition, is packaged into a *behavior*. Behaviors play the role of situated actions: they indicate which movement should be performed under which circumstances and with respect to which objects. Behaviors bridge the gap between abstract action descriptions and physical control. They are inherently movement-oriented, since their control schema induces a preference on control actions; but they also incorporate elements of abstract action: the objects that the action operates on, and the preconditions for the success of the action.

Our final step is to link the local notions of sensing and control embodied in behaviors to the global notion of satisfaction of goals. To do this, we adapt some ideas from dynamic logic to our multivalued framework. We define the meaning of a behavior to be the (fuzzy) set of all possible executions that it can produce, expressed as trajectories in the space of states. Then, goals can be defined as predicates on the trajectories. We relate behaviors to goals by defining the central notion of *goodness* of a behavior for a goal: any execution of the behavior under its contextual condition will satisfy the goal. For example,

suppose $B(a)$ is a behavior for picking up a block a. We can show that it accomplishes this goal under circumstances C if the predicate $Grasping(a)$ is true of every trajectory of $B(a)$ in every environment where C is true. In fact, this is a uniform way to describe not only goals of achievement, but also goals of maintenance and prevention.

Behaviors can be combined to form complex controllers. If we could prove that the composition of behaviors accomplishes a corresponding composition of their goals, we could use techniques for reasoning about actions and goals to generate complex controllers. In fact, under certain hypotheses of consistency and stability, we are able to prove two theorems that relate composed behaviors to composed goals. First, if two behaviors are individually good for two goals G_1 and G_2, then their *conjunction* is good for the conjunctive goal $G_1 \cap G_2$. Second, if a behavior B is good for G under the circumstances C, and B' is good for bringing C about whenever in C', then the *chaining* of B and B' is good for G under the more general circumstances $C \cup C'$.

We now have all the element we need to reason about behaviors and their composition. Behaviors bring with them a specification of their preconditions (the context) and their postconditions (the goals they are good for); and they can be combined with classical composition operators, like conjunction and chaining. Thus, we can reasonably expect to be able to use standard planning techniques, and existing planning systems, to generate complex combinations of behaviors that satisfy given goals. This is indeed the case, and we end the paper by showing how we can use use deliberation processes based on goal regression and goal reduction to generate complex controllers. We emphasize that we do not propose here any novel planning technique: our aim is to show that multivalued controllers can be generated by the existing ones.

1.2 Experiments on a mobile robot

We have tested the proposed methodology extensively on indoor robot navigation problems, using a mobile robot testbed called Flakey. To make our approach more concrete, we use this testbed throughout the paper, by showing runs and discussing implementations choices. Our experience on Flakey indicates the effectiveness of the discussed methodology, and the ease of incorporating different types of deliberation processes.

1.3 Plan of the Paper

The next section discusses our multivalued logic approach to define complex controllers, and introduces the notions of control schema, context, trajectory,

and context-depending blending. In Section 3, we consider the perceptual embedding of the agent, and define behavior schemas and their three combination operators: conjunction, blending, and chaining. Section 4 completes the formal construction by introducing the notions of a goal and of goodness of a behavior for a goal, and presenting the composition theorems. In Section 5, we exploit the results of the previous sections and show how we can reason about behaviors and generate complex controllers using standard planning techniques. Finally, Section 6 compares our method to other related approaches. An extended version of this paper is available as [SKR93].

2 Moving

We start our construction from the description of the physical movements of an agent. We describe elementary movements, like stretching an arm or rolling forward, by structures called *control schemas*. Then we turn our attention to the execution of movements, and study the set of all possible executions, or *trajectories,* of a control schema. Finally, we show how to compose multiple control schemas to describe complex movements, and give some theorems that relate the trajectories of a combined control schema to those of its components. To make things more concrete, we illustrate the formal definitions by examples taken from our mobile robot, Flakey. First, some background on the formal tools that we use.

2.1 Multivalued logics

Throughout this paper, we use the framework of multivalued logics [LT83,Res69]. There are two main reasons for this choice. The first one is technical: by representing degrees of truth on a numeric scale, multivalued logics provide an ideal framework to merge notions taken from the world of planning, typically expressed in symbolic terms, with notions taken from the world of control, typically expressed in numeric terms. The second reason is semantic. Multivalued logics can be viewed as logics of graded preference [Res67] where we interpret the truth value of a proposition P in a world as the utility, or *desirability*, of being in that world from the point of view of P [BZ70,Rus91a,Rus91b]. Accordingly, we use propositions to represent control strategies and goals, and logical connectives to combine them.

Let $\mathcal{L}$ be a propositional language, S a set of states, and $\mathsf{TV} : \mathcal{L} \times S \to [0,1]$ a function that assigns a truth value in $[0,1]$ to each atomic proposition in each

state. We extend TV to non-atomic propositions by the equations

$$\begin{aligned}
\mathsf{TV}(\neg P, s) &= \quad 1 - \mathsf{TV}(P, s) \\
\mathsf{TV}(P \wedge Q, s) &= \mathsf{TV}(P, s) \otimes \mathsf{TV}(Q, s) \\
\mathsf{TV}(P \vee Q, s) &= \mathsf{TV}(P, s) \oplus \mathsf{TV}(Q, s) \\
\mathsf{TV}(P \rightarrow Q, s) &= \mathsf{TV}(Q, s) \oslash \mathsf{TV}(P, s)
\end{aligned} \tag{1}$$

where $\otimes$ is any continuous triangular norm, or *t-norm*, with quasi-inverse $\oslash$, and $\oplus$ is any continuous *t-conorm*.

T-norms, the quasi-inverse, and t-conorms are used as a generalization of logical conjunction, implication, and disjunction, respectively [SS83,Web83,Val85]. Mathematically, a t-norm is any binary operator on $[0, 1]$ that is commutative, associative, non-decreasing in each argument, and has 1 as unit; a t-conorm has the same properties but has 0 as unit. Given a continuous t-norm $\otimes$, its quasi-inverse $\oslash$ is defined by

$$x \oslash y = \sup\{\omega \in [0, 1] \mid \omega \otimes y \leq x\}\,.$$

Note that, for any $\otimes$, $y = 0$ implies $x \oslash y = 1$ and $y = 1$ implies $x \oslash y = x$, which justifies the use of $\oslash$ to model logical implication.

Example 1 *The following are examples of t-norms and t-conorms.*

	$x \otimes y$	$x \oplus y$	$x \oslash y$
a)	$\min(x, y)$	$\max(x, y)$	$\begin{cases} 1 \text{ if } x \geq y\,, \\ x \text{ otherwise} \end{cases}$
b)	xy	$x + y - xy$	$\min(x/y,\, 1)$
c)	$\max(x + y - 1,\, 0)$	$\min(x + y,\, 1)$	$\min(x - y + 1,\, 1)$

The operators in (a) are the ones we use in our robot, Flakey. Notice that if we restrict truth values to be either 0 or 1, the conditions (1) reduce to classical logic for any choice of $\otimes$, $\oplus$ and $\oslash$.

Given a proposition P, we are often interested in the set of states of S where P holds, denoted by $[\![P]\!]$. As truth values are numbers in $[0, 1]$, the membership of any state s to $[\![P]\!]$ is a matter of degree. We measure this degree by the membership function $f_{[\![P]\!]}$ defined by

$$f_{[\![P]\!]}(s) = \mathsf{TV}(P, s)\,.$$

Graded sets of this type are commonly known as *fuzzy sets* [Zad65]. For notational simplicity, and when there is no risk of ambiguity, we denote the fuzzy set $[\![P]\!]$ simply by P, and write $P(s)$ for $f_{[\![P]\!]}(s)$. We define complement, intersection and union of fuzzy sets based on the equations (1) above as follows:

$$\begin{aligned} \overline{P}(s) &= 1 - P(s) \\ (P \cap Q)(s) &= P(s) \otimes Q(s) \\ (P \cup Q)(s) &= P(s) \oplus Q(s) \end{aligned} \tag{2}$$

We also define set inclusion by

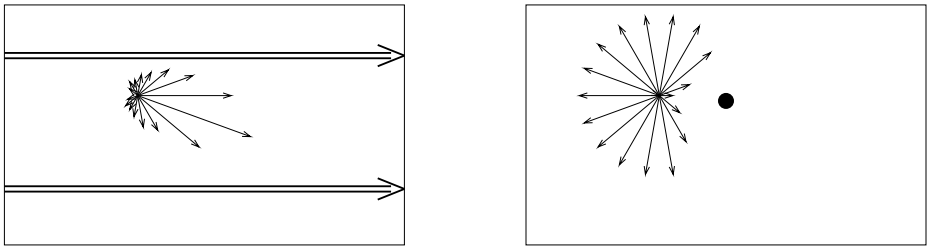
$$P \subseteq Q \text{ iff } P(s) \leq Q(s) \text{ for all } s \text{ in } S.$$

If we use the operators (a) in Example 1, these operations correspond to those originally defined by Zadeh [Zad65].

In what follows, we sometimes refer to a proposition P as a *desirability function* on S to emphasize the interpretation of $P(s)$ as the desirability of being in state s from the viewpoint P. We routinely use the $\otimes$, $\oplus$ and $\oslash$ operators to combine desirability functions. The reader should keep in mind that $\otimes$ is meant to capture the notion of conjunction, $\oplus$ the notion of disjunction, and $\oslash$ the notion of implication (right to left). We also use the sup (least upper bound) and inf (greatest lower bound) operators to capture the notions of existential and universal quantification, respectively.

2.2 Control schemas

Consider an agent with a set S of possible internal states, and a set A of (atomic) control actions. A control strategy for this agent is generally defined by a function that produces, in each state, a control action. To define generic types of movements — movements that can be instantiated in several ways depending on the circumstances of execution — we need to be less rigid. For concreteness, consider the type of movement "take a step." We may well have an archetype of the ideal step, but when we take a step on a muddy road we may perform a movement that only vaguely meets this archetype; still, this inelegant maneuver is probably the best choice available in that situation, and we want our definition of "take a step" to account for it. We extend the notion of a control strategy to produce, in each state, a value of *desirability* of each possible control.



(a) (b)

Fig. 1. Values of $D(s, a)$ for two control schemas: (a) Follow; (b) Keep-Off.

Definition 2 *Let S be a set of states, and A a set of control actions. A* control schema *on S and A is any function*

$$D : S \times A \to [0, 1] .$$

Intuitively, the value of $D(s, a)$ measures how much executing a in state s is desirable from the viewpoint of performing that type of movement.

Example 3 *Figure 1 illustrates two control schemas. In (a), we consider a control schema, called* Follow*, for proceeding within a given "lane" (represented by the double lines); (b) refers to the* Keep-Off *control schema, intended to stay away from a given spot. The picture is a representation of the current state s, with the agent being located at the small dot. Each vector indicates a possible turning direction a: the length of the vector is proportional to the desirability $D(s, a)$.*

Note that the control actions in A are meant to be directly executable low-level commands. In the case of our robot Flakey, the elements of A are the turning and velocity control values to be sent to the wheel effectors at each control cycle (100 msec).

2.3 Trajectories

A control schema is a local notion: it tells us the desirability of control actions at each state. In order to get a more global view of movements, we introduce the concept of a *trajectory*. This notion is essential in relating movements to goals in Section 5.

Definition 4 *Let S be a set of states. A* trajectory *on S is a finite, nonempty sequence of elements of S. The set of all trajectories on S is denoted by $\mathcal{T}(S)$. We also denote by $\mathcal{T}_n(S)$ the set of trajectories of length at least n.*

We take trajectories on S to represent complete executions of movements, and (fuzzy) sets of trajectories to represent *types* of movements. For example, an agent's hand can follow an infinite number of paths when the agent stretches its arm, and all these trajectories are instances of the type of movement "stretch an arm." Accordingly, we characterize the type of movement defined by a control schema D by the set of its possible executions, that is, the set of trajectories produced by always executing control actions that are desirable according to D.

Formally, we proceed as follows. First, we assume that we are given a function

$$M : S \times A \times S \to [0,1]$$

such that $M(s,a,s')$ measures how possible it is that s' is the state resulting from the execution of action a in state s. In general, we expect M to account for the agent's physical capabilities and constraints, for the known environmental features, for the uncertainty associated with effector failures or interference of other agents, and so on. For example, robots have bounded acceleration, so any two successive states in which the velocity increases too much cannot be in an admissible trajectory. Obviously M can be a very complicated relation, especially for nonholonomic robots; its main job here is to allow us to abstract from the details of robot motion.

Given a control schema D, we measure the possibility that an execution of D produces a transition from a state s to a state s' by

$$Next_D(s,s') = \sup_{a \in A} \left[D(s,a) \otimes M(s,a,s') \right]. \tag{3}$$

Recalling our intended reading of the sup and $\otimes$ operators, we can interpret (3) as saying that (s,s') is a possible state transition for D to the extent that there exists some action a that is both desirable for D in s, *and* it leads from s to s'.

Finally, we look at the set of trajectories whose transitions are all possible for D. We call these trajectories *desirable* for D.

Definition 5 *Let D be a control schema. The set of* desirable trajectories *of D is the fuzzy set on $\mathcal{T}(S)$ defined by*

$$Traj_D(t) = \inf_{0 \le i < k} Next_D(s_i, s_{i+1}).$$

where $t = (s_0, s_1, \ldots, s_k)$, $k > 0$.

The set $Traj_D$ fully characterizes a control schema D, and we take this set to be the *meaning* of D. This is similar to the approach taken in dynamic logic, where a program is associated with the set of its possible executions. Segerberg [Seg85] and Israel, Perry and Tutiya [IPT91] also analyze movements and actions in similar terms.

2.4 Implementation of control schemas

Let us now pause for a moment and see how control schemas relate to the acutal control of a physical agent. A control schema is a descriptive device. Any implementation of a control schema D must select, at each state, one control action to send to the effectors. We represent such an implementation by a control function

$$F_D : S \to A\,.$$

Selecting actions by the F_D function corresponds to choosing and following one of the desired trajectories, i.e., to execute one particular movement of type D. Of course, we would like to choose the "best" trajectory. This is an intractable search problem, as the size of the state space grows exponential with its dimensionality. Similar problems occur with potential field methods (see Section 6). There are many different approximate search methods that could be employed to find candidate trajectories. We have built an implementation of control schemas for our mobile robot Flakey by using techniques based on fuzzy control [Rus90,SRK93b,SRK93a]. A control schema D is encoded by a set R of fuzzy rules of the form

$$\text{IF } P_i \text{ THEN } A_i\,,\quad i = 1, \ldots, n\,,$$

where each P_i is a proposition in multivalued logic, and each A_i is a fuzzy set of control actions. From these fuzzy rules, a desirability function D_R can be computed by

$$D_R(s,a) = \sup_{1 \le i \le n} \min(P_i(s), A_i(a))\,. \tag{4}$$

Intuitively, D_R says that a is a desirable control in s if there is some rule in R that supports a and whose antecedent is true in s. This way to interpret control rules is customary in the field of fuzzy control. The process of choosing one action from this desirability function is called *defuzzification*. We have used

IF (centerline-on-right $\wedge$ $\neg$lane-angled-left) THEN turn-medium-right

IF (centerline-on-left $\wedge$ $\neg$lane-angled-right) THEN turn-medium-left

IF (lane-angled-right $\wedge$ $\neg$centerline-on-left) THEN turn-smooth-right

IF (lane-angled-left $\wedge$ $\neg$centerline-on-right) THEN turn-smooth-left

Fig. 2. Fuzzy rules used in Flakey to implement the Follow schema.

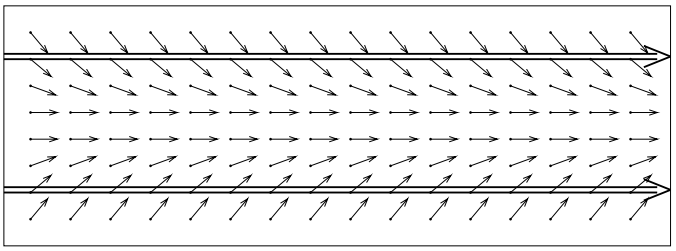


Fig. 3. A vector field showing the controls generated by the Follow ruleset.

the centroid method, which computes a desirability-weighted average control

$$F_R(s) = \frac{\int a\, D(s,a)\, \mathrm{d}a}{\int D(s,a)\, \mathrm{d}a}. \tag{5}$$

For averaging to make sense, the rules should not suggest dramatically opposite actions in the same state. Our coding heuristic has been to make sure that rules with conflicting consequents have disjoint antecedents. Other authors have preferred to use more involved choice functions (e.g., [YP92]).

Example 6 *Figure 2 shows Flakey's rules for implementing the* Follow *control schema discussed above. These rules keep the robot close to the middle of the lane (first two rules), and approximately lined up with it (last two). Figure 3 shows the turning controls produced by the corresponding F_R. Note that, in each state, F_R only outputs one turning control, and does not say anything about the desirability of alternatives — contrast this with the desirability function pictured in Figure 1(a).*

Flakey's fuzzy rules implement a local "greedy" method, gradient descent, that is highly reactive and simple to compute. The fuzzy rule format makes it easy to write and debug simple control schemas, and it should also make it easier to learn or improve schemas automatically. We have written rulesets for a dozen control schemas, including ones for avoiding an obstacle, for crossing a door, for reaching a near location, and so on. It is the responsibility of higher-level methods to only instantiate schemas for which gradient descent is appropriate based on some global analysis, or to monitor schemas to detect local minima and failure of the schema (see Section 5).

An agent is often engaged in activities that require the simultaneous activation and coordination of several control schemas. For instance, a robot may be going down a hallway while avoiding obstacles and moving its camera to keep track of some landmark. The resulting overall movement can be characterized by an appropriate combination of control schemas.

Two control schemas D_1 and D_2 can be combined in several ways. The most basic one is the *conjunctive combination*: at each state, we consider actions that are desirable for both D_1 and D_2. Formally, we simply intersect the fuzzy sets D_1 and D_2:

$$(D_1 \cap D_2)(s,a) = D_1(s,a) \otimes D_2(s,a)\,. \tag{6}$$

The *disjunctive combination* of D_1 and D_2 is defined in a similar way using $\cup$ and $\oplus$. As it may be expected, the desirable trajectories of the conjunction are at most those of each conjunct; and those of the disjunction are at least those of each disjunct.

Theorem 7 *Let D_1 and D_2 be two control schemas. For any trajectory t in $\mathcal{T}(S)$,*

(i) $Traj_{D_1 \cap D_2}(t) \leq Traj_{D_1}(t)$;
(ii) $Traj_{D_1 \cup D_2}(t) \geq Traj_{D_1}(t)$.

Conjunctive combination works well for combining control schemas that are not conflicting; that is, in every state there is some control that is desirable for both schemas. When they conflict, the combined desirability function would be identically zero in that state. Unfortunately, this is more the normal case than the exception. For example, we may want to combine a control schema for going down the middle of a corridor, and one for staying away from obstacles. In the state where the robot is facing an obstacle, the first schema would prefer controls that go forward, while the second one would favor controls that turn the robot, say, right. The key observation here is that each control schema has in general its own *context* of applicability, and each desirability function should be considered only when appropriate. In the previous example, the first control schema can be sensibly applied only in situations where the space in front of the robot is free. When the obstacle is detected, this schema is outside its area of competence, and we should disregard its desires.

Formally, we restrict the area of competence of a control schema to a context as follows.

Definition 8 *Let S be a set of states. A* context *on S is a fuzzy set of S.*

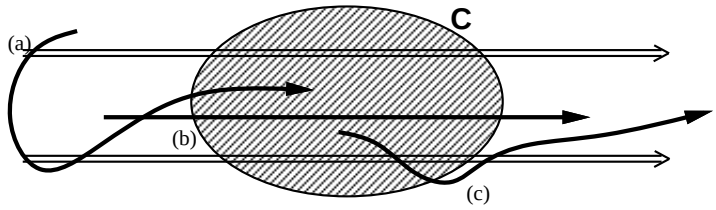


Fig. 4. The restriction of the Follow control schema to the context C. Trajectories (a) and (b) are both desirable; (c) is not desirable.

The notion of context C on S may be extended to a corresponding structure in the set of trajectories $\mathcal{T}(S)$ in a natural way, i.e., a trajectory is as desirable as its least desirable state.

$$C(t) = \inf_{s \in t} C(s)\,.$$

Definition 9 *Let D be a control schema on S and A, and C a context on S. The* context restriction *of D to C, denoted $D^{\downarrow C}$, is the control schema defined by the expression*

$$D^{\downarrow C}(s,a) = D(s,a) \oslash C(s)\,.$$

By using our interpretation of $\oslash$, we read $D^{\downarrow C}$ as saying that *if* a state s is in the context C, *then* the desirability of a control a is given by $D(s,a)$; otherwise, any control is admissible. Mathematically, we have $D^{\downarrow C}(s,a) = D(s,a)$ if $C(s) = 1$, and $D^{\downarrow C}(s,a) = 1$ if $C(s) = 0$. Hence, action must obey D in the states inside C, and is free outside C. Correspondingly, the desirable trajectories for $D^{\downarrow C}$ are those that obey D inside C, and are totally free outside, as shown by the following theorem.

Theorem 10 *Let D be a control schema and let C be a context on S. If t is a trajectory, then it is true that*

$$Traj_{D^{\downarrow C}}(t) \leq \inf_{t' \subseteq t} \left[Traj_D(t') \oslash C(t') \right],$$

where $t' \subseteq t$ means that t' is a subtrajectory of t.

That is, for t to be desirable for $D^{\downarrow C}$ it is necessary that all its segments inside C be desirable for D.

Example 11 *Figure 4 shows a lane and three trajectories. Consider the control schema* Follow$^{\downarrow C}$. *Trajectories (a) and (b) are both desirable for* Follow$^{\downarrow C}$, *as they run inside the lane whenever in C; trajectory (c) is not desirable, as*

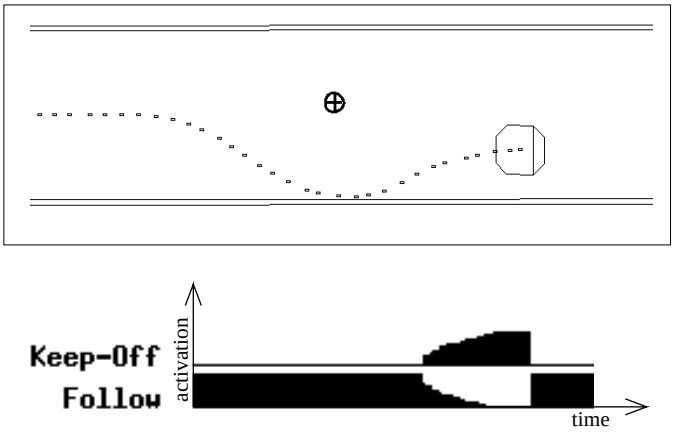


Fig. 5. Context-dependent blending of the Keep-Off and Follow control schemas.

it fails to follow the lane while in C. Note that trajectory (a) would not be desirable for the unrestricted Follow.

We can combine context restriction and conjunctive combination into a general combination pattern, called *context-dependent blending.* Each control schema is associated with a context, meant to identify the states where the schema is competent.

Definition 12 *Let D_1 and D_2 be two control schemas, and C_1 and C_2 two contexts. The* context-dependent blending *of D_1 and D_2 under C_1 and C_2 is given by:*

$$D_1^{\downarrow C_1} \cap D_2^{\downarrow C_2} \ .$$

2.6 Implementation of context-dependent blending

Context-dependent blending is implemented in Flakey by meta-level fuzzy rules of the type

$$\text{IF } C_j \text{ THEN } R_j,\ j = 1, 2, \ldots, m\,,$$

where each R_j is a ruleset implementing a control schema, and C_j is a context for R_j. Given a current state s, Flakey's controller computes the function D_{R_j} for each R_j according to equation (4) above. All the D_{R_j}'s functions are then combined modulo the C_j's contexts, according to Def. 12, and the resulting tradeoff function D_{tot} is fed to (5) to produce the control value $F_{\text{tot}}(s)$.

Example 13 *Figure 5 shows an example of context-dependent blending of the* Follow *and the* Keep-Off *control schemas. The meta-rules used to encode the*

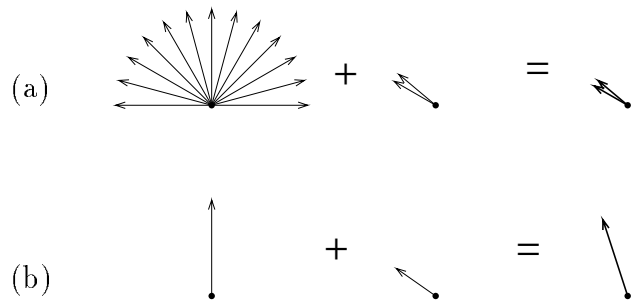


Fig. 6. Combination of full desirability functions (a) and of their representatives (b).

contexts are:

IF spot-close-in-front THEN Keep-Off
IF ¬spot-close-in-front THEN Follow

The upper part of the picture is a geometrical representation of the internal state of Flakey at one point during execution. Flakey is drawn in top-view heading right. The double lines and the crossed circle represent the locations of the lane to follow and of the spot to avoid. The lower part of the picture plots the evolution over time of the truth value of the contexts, hence the level of activation of the corresponding control schemas in the blending. The trajectory executed by Flakey is desirable for the blending — in particular, it is desirable for the restriction of Follow *to the context* ¬spot-close-in-front*, as it leaves the center of the lane only when the context is false.*

It is important to emphasize that an implementation of context-dependent blending should *first* combine the component desirabilty functions, effectively forming a full preference function, and *then* chose one preferred control action from the combined function. The distinction can be seen in the following example (Figure 6). In (a), the desirability function D_1 strongly prefers any direction from $-90°$ to $90°$; D_2 weakly prefers a narrow angle from $50°$ to $60°$. Their combination is dominated by D_2, since D_1 is indifferent over such a large range. Suppose now that we first summarize each desirability function by choosing one preferred control, and then combine these controls (b). Notice the result is very different from (a), and does not represent the best choice for movement. The context-dependent blending mechanism constitutes the main technical difference between our control schemas and other methods based on a weighted combination of local preferences (e.g., the potential field methods discussed in Sect. 6 below).

3 Behaving

In this section, we study how control schemas can be "lifted" from the level of movements to the level of behaviors in an environment. Behaviors correspond to the concept of situated actions, and are normally associated with the accomplishment of goals. For example, the behavior of "following this wall" can be a way of reaching any one of the doors in that corridor, or a way of following a person. In this section we concentrate on developing the structure of behaviors, and the ways behaviors can be combined, or *blended,* together. We also touch at issues of relating behaviors to perception. In the next section we relate behaviors to goals.

3.1 Behavior schemas

Our approach to lifting a specification of a type of movement to a specification of a behavior in an environment is by associating it with a context of execution and with a set of objects to operate on. For example, consider the movement type "extending the right arm." If executed in the direction of a cup, and in a situation in which the cup is within the arm's reach, this movement will result in the production of a behavior of type "hit a cup" (which in its turn may be a way of achieving a goal like "break a cup" or "win a prize.") In general, we characterize a type of behavior by a *behavior schema*: a specification of *what movement* should be performed with respect to *which objects* and under *what circumstances.*

Definition 14 *Let S be a set of (internal) states, and A a set of control actions. A* behavior schema *on S and A is a triple*

$$B = \langle C, D, O \rangle$$

where C is a context on S, D is a control schema on S and A, and O is a set of object descriptors.

The new element in this definition is the set of object descriptors. The control schema D and the context C operate on formal variables in the internal state, e.g., the direction for moving the arm. If the agent has to act with respect to an external object, these variables must reflect properties of that object, e.g., the position of the cup. We group the variables related to an object into a partial model of the object, called an *object descriptor.*[2] Object descriptors

[2] Note that each descriptor is specific to a behavior: it should include all and only the properties of the object that are relevant to that behavior.

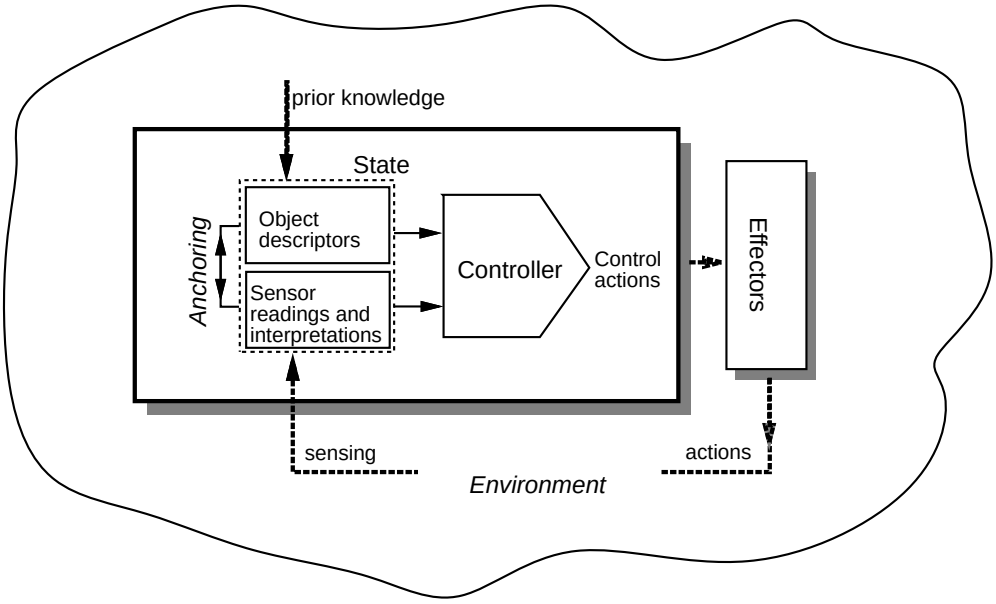


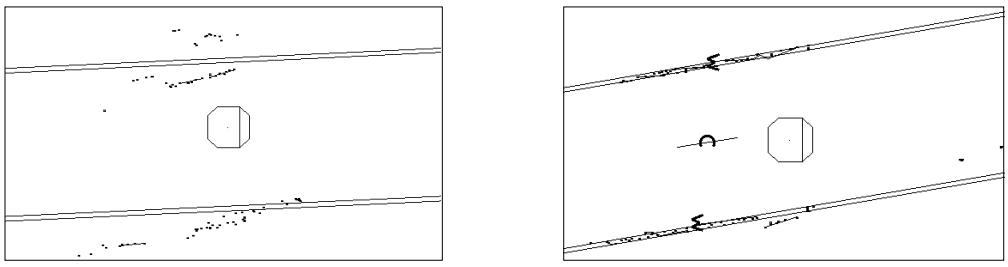
Fig. 7. Our approach to embedding a controller in the environment.

are essential to link abstract specifications of actions to physical execution (see Fig. 7). Suppose we want to execute the action "pick up cup A." The identifier "cup A" does not have any meaning to the controller. So, we create a descriptor for this cup based on properties stored in the agent's long-term memory (e.g., a map), and pass this descriptor to the controller. The controller then operates in an open-loop fashion with respect to the properties in the descriptor. Later, when the actual cup is perceived, we use the output from the perceptual system to continuously update the descriptor, effectively switching to a closed-loop regime. We refer to the process of keeping object descriptors coordinated with physical objects through perception as *anchoring* [Saf94].

Example 15 *The* Follow *control schema introduced in Section 2.4 can be used to build a behavior schema for moving down a given corridor, provided that Flakey is in the corridor and it is not blocked by an obstacle:*

$$\langle\, [\mathsf{at}(\mathsf{Corr1}) \wedge \neg \mathsf{facing}(\mathsf{OG})],\ \mathsf{Follow}(\mathsf{Corr1}),\ \{\mathsf{Corr1},\ \mathsf{OG}\}\ \rangle\,,$$

where we indicate in parentheses a dependency on (the properties of) an object descriptor. The Corr1 *descriptor includes a lane that approximates the size and position of the actual corridor we want to follow.* OG *denotes the "Obstacle Grid," a special descriptor used for obstacle avoidance that matches any object around Flakey. The context C is expressed in a logical form: in any state s, the value of $C(s)$ is computed through simple geometrical reasoning, and from equations (1). Fig. 8 shows the internal state of Flakey while executing this behavior. Note that the state now includes the input from the sensors and the*



(a) (b)

Fig. 8. Anchoring a corridor descriptor to sensor readings for corridor-following.

perceptual interpretations built from this input. In the picture, each small dot represents a sonar reading, indicating that something has been detected at that spot, and short segments indicate surfaces reconstructed from these readings. Initially (a) the position of the lane is set accordingly to an internal map. When enough sonar readings are gathered, Flakey's perceptual routines infer the existence of two parallel walls, marked by "W", and a corridor, marked by "C". This information is used to update the Corr1 *descriptor (b), and Flakey's motion now follows the actual corridor.*

The last example illustrates the role of anchoring in going from a type of movement, "go straight between two boundaries," to a type of behavior, "follow a corridor," by relating the two boundaries to the perceived walls. Note that executing the same type of movement on a road and anchoring the two boundaries to the perceived white lines would result in a behavior of type "drive in a traffic-lane." (See [MS90] for a related approach to *encapsulating* behavior.)

For each descriptor d, we assume we have a function

$$Anch_d : S \to [0,1]$$

such that $Anch_d(s)$ measures the extent to which d is anchored in state s. In most practical cases, $Anch_d$ will be a binary function that returns either 0 or 1. *Anch* is extended to sets O of descriptors and to trajectories in the obvious way:

$$Anch_O(s) = \inf_{d \in O} Anch_d(s)$$

$$Anch_O(t) = \inf_{s \in t} Anch_O(s)\,.$$

We are now ready to define the *meaning* of a behavior. As we did for control schemas, we identify the meaning of a behavior schema B with the set of all trajectories that can result from executing B under the correct anchoring. We call these trajectories *admissible.*

Definition 16 *Let t be a trajectory in $\mathcal{T}(S)$, and $B = \langle C, D, O \rangle$ a behavior schema. The degree by which t is an* admissible execution *of B is given by:*

$$Adm_B(t) = Traj_{D \downarrow C}(t) \otimes Anch_O(t).$$

That is, a trajectory t is admissible for behavior B if the descriptors O are anchored in t, and t is desirable for D whenever in the context C. By virtue of the anchoring of the object descriptors, this trajectory in the formal domain S corresponds to an actual trajectory in the environment. This means that we can analyze the properties of a behavior schema by looking at its admissible trajectories Adm_B.[3]

We may need to know if a behavior has admissible trajectories that lie in its context. Often, we even require that the behavior has arbitrarily long such trajectories. We measure the *satisfiablility* of a behavior by the following function (recall that $\mathcal{T}_n(S)$ denotes the set of trajectories of length at least n).

$$Sat(B) = \inf_{n \geq 0} \sup_{t \in \mathcal{T}_n(S)} \left[Adm_B(t) \otimes C(t) \right]. \tag{7}$$

3.2 Blending behaviors

Complex behaviors can be created by composing, or *blending,* basic behaviors, using the context-dependent blending of desirability functions defined in Section 2. We distinguish three different flavors of blending, depending on how the contexts are used in the combination.

Definition 17 *Let $B_1 = \langle C_1, D_1, O_1 \rangle$ and $B_2 = \langle C_2, D_2, O_2 \rangle$ be two behavior schemas. Then the following are behavior schemas:*

[3] The situation is more complex. Anchoring does not link object descriptors with external objects, but with their perceptual images inside the agent. These images reflect the properties of the actual objects only as far as the perceptual apparatus is reliable. Accounting for the reliability of perception is a difficult issue that lies beyond the scope of this paper.

$$\text{CONJ}[B_1; B_2] = \langle\ C_1 \cap C_2,\ D_1 \cap D_2,\ O_1 \cup O_2\ \rangle$$

$$\text{BLEND}[B_1; B_2] = \langle\ C_1 \cup C_2,\ D_1^{\downarrow C_1} \cap D_2^{\downarrow C_2},\ O_1 \cup O_2\ \rangle$$

$$\text{CHAIN}[B_1; B_2] = \langle\ C_1 \cup C_2,\ D_1^{\downarrow C_1 \setminus C_2} \cap D_2^{\downarrow C_2},\ O_1 \cup O_2\ \rangle\ .$$

The conjunctive operator CONJ provides the simplest form of combination: it builds a more focused behavior schema that considers two control schemas simultaneously in their common context. The BLEND operator builds a behavior schema by applying each component behavior in its own context. And the chaining operator CHAIN is a special case of blending where the second behavior takes priority over the first one in the common part of the context.

Composite behaviors are implemented in Flakey using the mechanism for context-dependent blending of control schemas. In fact, by using min for $\otimes$, any combination of desirability functions produced by the CONJ, BLEND and CHAIN operators can be expressed in the canonical form

$$D_1^{\downarrow C_1} \cap D_2^{\downarrow C_2} \cap \ldots \cap D_n^{\downarrow C_n}\ , \tag{8}$$

and then be implemented by meta-rules as shown in Example 13 above.

Example 18 *The simple corridor following behavior defined in Example 15 assumed that there was no obstacle in front of Flakey. We can blend this behavior with the obstacle avoidance behavior built around the* Keep-Off *control schema to obtain a composite behavior to go down a corridor while avoiding obstacles on the way:*

BLEND[⟨ at(Corr1) ∧ ¬facing(OG), Follow(Corr1), {Corr1, OG} ⟩ ;
⟨ facing(OG), Keep-Off(OG), {OG} ⟩]

where Corr1 *and* OG *are as in Example 15. Notice that the blended behavior applies in the context of being* at(Corr1)*, irrespective of there being any obstacles, that is, in a wider context than the* Follow *behavior alone. This blending is implemented by the following two meta-rules:*

IF facing(OG) THEN Keep-Off(OG)
IF at(Corr1) ∧ ¬facing(OG) THEN Follow(Corr1)

Figure 9 shows a run of this composite behavior on Flakey. (The Sense *behavior will be discussed in the next subsection.) (1) shows Flakey's internal state at moment (d). (2) shows an external view of the environment where Flakey moves. (3) plots the level of activation of the three control schemas over time. Spikes in the activation levels are caused by noisy sonar readings.*

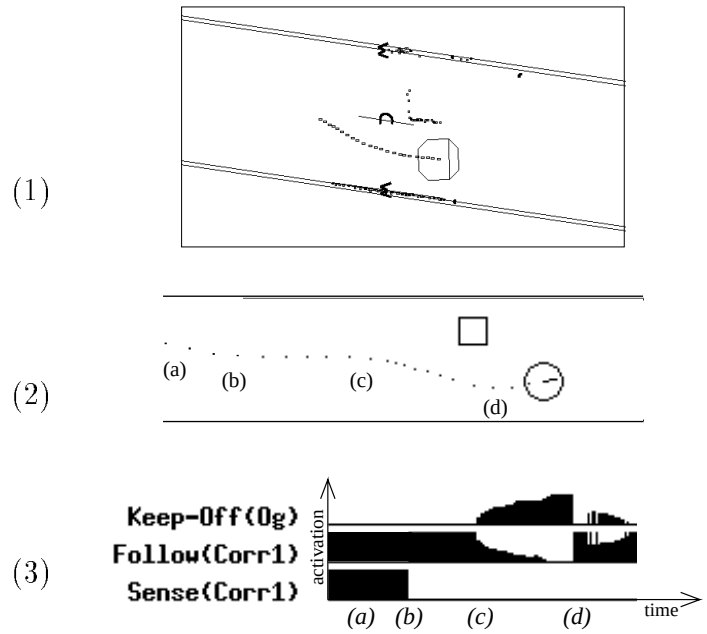


Fig. 9. Context-dependent blending of corridor-following and obstacle-avoidance.

Instants (a) and (b) correspond to the two snapshots of Fig. 8 above. Notice the new heading taken by Flakey after the corridor has been anchored (b). In (c), the obstacle has been detected by Flakey's sonars, and anchored to OG. *Hence, the preferences of* Keep-Off *begins to take over those of* Follow. *Later, when the path is clear,* Follow *resumes full importance (d), and guides Flakey toward the center of the hallway. The overall pattern of control is similar to the one discussed in Example 13; however, through the anchoring of the internal variables used by the control schemas, this control now produces an effective behavior in the environment.*

Composition of behaviors plays a central role in our construction, and we will show how we can map plans generated in some standard way into composite behaviors, and hence into embedded controllers. However, not every composition makes sense. In particular, one should be careful that the desirability functions of two behaviors being combined be mutually consistent — i.e., they do not conflict in the same context. We measure the mutual consistency of two behaviors B_1 and B_2 by the value of $Sat(\text{CONJ}[B_1; B_2])$. That is, B_1 and B_2 are mutually consistent if we can find trajectories arbitrarily long that are admissible for both behaviors in their common context.

3.3 Behavior and perception

The object descriptors constitute a model of the real-world objects that are relevant to the control task: the anchoring process keeps them coordinated with the input coming from the perceptual system. In practice, we have found

that the introduction of object descriptors greatly simplifies the design of behaviors, by allowing us to decouple the problem of control from the problems of interpreting noisy sensor data. Our behavior-writing methodology in Flakey has been to first write small rulesets for elementary types of movements based on simple descriptors, like follow a line, or reach a location; and then focus on the strategies to keep these descriptors anchored to the right features in the environment. The resulting behaviors often proved to be more robust than purely reactive controllers. For example, our wall following behavior can produce useful movement even when the actual wall is temporarily obscured. Intuitively, the controller follows the generic direction marked by the wall, rather than the actual contour of the wall, and this direction is registered from time to time through perception. Notice that anchoring is normally recomputed many times during action execution (possibly at every control cycle), providing a closed-loop response whenever the relevant sensor data are available. When data are not available, e.g. when starting the behavior, the descriptors act as assumptions originating from the last anchoring, or from prior knowledge.

There are several places where the introduction of the object descriptors and of the anchoring mechanism helps us to better understand the points of contacts between perception and action. A first such point of contact is possibility to focus the perceptual system. Gathering a full geometric picture of all the objects in the immediate environment is time consuming and impractical given current computational limitations, and leads to slow reactions in dynamic situations. However, particular behaviors may need only part of the full perceptual information available. As the object descriptors list all the perceptual properties that are relevant to a given behavior, their content can be examined by the perceptual routines to focus attention on those features of the environment that are relevant to the behavior. A similar approach has been adopted by Arkin in his perceptual schemas [Ark87,Ark90].

A second point of contact is the need of some behaviors to keep track of the anchoring of their objects. For example, a behavior for performing some precise manipulation on a workpiece should make sure that the workpiece has been perceived and anchored. This can be done by including the value of $Anch_O$ in the context. More interestingly, behaviors may actively try to achieve and maintain the anchoring. One way to do this is by issuing commands to the perceptual apparatus, e.g., turning a camera or activating an interpretation routine. Another way is by preferring movements that are helpful to the perceptual process.

Example 19 *We discuss the role of the* Sense *behavior in Example 18 above. It is easier for perception to find the corridor walls if the robot moves slowly along the corridor in a linear fashion, without turning. Over several seconds, the sonar sensors will approximate a long synthetic aperture, and generate a reliable reading of the wall location. The* Follow *control schema is designed to*

move fast along the midline of the corridor: under some conditions (reflective or discontinuous walls), this may make wall recognition more difficult. To remain anchored in these situations, the Follow *schema is blended with the following one:*

$$\langle \neg \mathsf{anchored}(\mathsf{Corr1}), \mathsf{Sense}(\mathsf{Corr1}), \{\mathsf{Corr1}\} \rangle .$$

When the corridor descriptor is still unanchored, as in Fig. 9(a), the Sense *control schema blends in and slows down the robot. Once the corridor is recognized by the perceptual subsystem, as in Fig. 9(b), the context of* Sense *becomes false, and the* Follow *behavior can proceed freely. Notice that if* Corr1 *later becomes unanchored, e.g., because a wall is occluded by obstacles, the* Sense *behavior will become active again and help Flakey to recover anchoring.*

In complex situations, simple strategies like the one above may not work, and higher-level decision-makers must be invoked to figure out how to recover the anchoring, or else form a new plan to achieve the desired goal (see Sect. 5). The advantage of using sensing behaviors that act to maintain anchoring is that they are simple and can be extremely reactive.

4 Goals

We now introduce *goals* as the final element of our construction, and study the relation between the execution of behaviors and the satisfaction of goals. The fundamental connection between goals and behaviors is formalized by the relation of *goodness*: a behavior is good for a goal if, in all trajectories produced by executing the behavior in its context, the goal holds. Additionally, we prove the main formal results of the paper, two composition theorems: the *conjunctive composition* of two behavior schemas is a complex control that satisfies the conjunct of their goals; and the *chaining composition* of two behavior schemas is a complex control that satisfies the goal of the second one in a wider context that includes the context of the first behavior. We will see in the next section how these results can be used for means-end reasoning and planning.

4.1 Representing goals

Definition 20 *Let S be a set of states. A* goal *on S is a fuzzy subset of the set of trajectories $\mathcal{T}(S)$.*

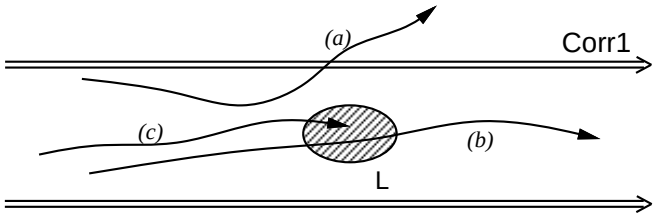


Fig. 10. Trajectories that satisfy different goals.

Given a goal G and a trajectory t, we read $G(t)$ as the degree by which t is a desirable trajectory for the goal. Specifying goals as sets of satisfactory executions is customary in control theory [Bel61], but is a less common in AI approaches to planning. One advantage of this choice is that both goals involving the *achievement* and goals involving the *maintenance* (or, for that matter, the *avoidance*) of some condition can be expressed in the same form. For example, if P and Q are (multivalued) predicates, we can define the following goals:

$$\text{ACHIEVE}[P](t) = \sup_{s \in t} P(s) \tag{9}$$

$$\text{MAINTAIN}[P](t) = \inf_{s \in t} P(s) \tag{10}$$

$$\text{ACHIEVE!}[P](t) = \sup_{0 \le i \le k} \inf_{i \le j \le n} P(s_j) \tag{11}$$

$$\text{SEQUENCE}[P,Q](t) = \sup_{0 \le i < n} \sup_{i < j \le n} [P(s_i) \otimes Q(s_j)] \tag{12}$$

where $t = (s_0, s_1, \ldots, s_n)$ is any trajectory in $\mathcal{T}(S)$. Intuitively, ACHIEVE$[P]$ is the goal of making P true: any trajectory that at some point makes P (partially) true is a (partially) good trajectory for ACHIEVE$[P]$. MAINTAIN$[P]$ is the goal of having P true all the time. ACHIEVE!$[P]$ requires that P be eventually true, and stay true until the end of the trajectory. And SEQUENCE$[P,Q]$ is the goal of making Q true after having made P true.

Example 21 *Figure 10 gives examples of trajectories that satisfy different goals. Trajectory (a) satisfies* ACHIEVE[at(Corr1)]*, but not* MAINTAIN[at(Corr1)] *or* ACHIEVE![at(Corr1)]*, because it leaves the corridor. (b) satisfies* MAINTAIN[at(Corr1)] *and* ACHIEVE[at(L)]*, but not* ACHIEVE![at(L)]*. (c) satisfies all of the three goals.*

It is useful to have a measure of a goal's *satisfiability*:

$$Sat(G) = \sup_{t \in \mathcal{T}(S)} G(t)\,, \tag{13}$$

meaning that there is some trajectory that can (partially) satisfy the goal.

We combine goals by means of the usual set-theoretic operators under the multivalued interpretation given in Section 2.1. For example, we can express the goal of a robot that wants to reach a certain location l_1 while avoiding another location l_2 by:

$$\text{ACHIEVE}[\text{at}(l_1)] \cap \text{MAINTAIN}[\neg\text{at}(l_2)] .$$

Conjunction of goals is a way to trade off several goals by preferring the trajectories that best satisfy all of them. The characteristics of this tradeoff depend on the particular t-norm employed. For example, if we use the min, we require that each of the conjuncts be satisfied at least at the level α in order for the conjunction to be satisfied at the level α; and if we use the product, we allow for a decrease in one conjunct to be compensated by an increase in the other one.

Finally, as we did for control schemas and behaviors, we introduce the notion of context restriction to focus a goal to a certain context — for example, the goal of avoiding obstacles only matters when there are obstacles nearby.

Definition 22 *Let G be a goal on S, and C a context on S. The* context restriction *of G to C, denoted $G^{\downarrow C}$, is the goal defined by*

$$G^{\downarrow C}(t) = \inf_{t' \subseteq t} [G(t') \oslash C(t')] .$$

In words, the goal $G^{\downarrow C}$ is satisfied by any trajectory that satisfies G whenever it is inside C (and moves freely outside).

4.2 Behaviors and goals

Executing a behavior may be a good way for satisfying some goals, and not others. In general a single behavior may satisfy more than one goal — for example, a wall-following behavior can achieve the goal of being at any given position in a corridor. We introduce the central notion of *goodness* to relate goals and behaviors. A behavior is good for a goal if any admissible trajectory of the behavior, in context, satisfies the goal.

Definition 23 *Let $B = \langle C, D, O \rangle$ be a behavior schema, and G a goal. The* goodness *of B with respect to G is defined by:*

$$Good(B, G) = \sup_{n>0} \inf_{t \in \mathcal{T}_n(S)} [G^{\downarrow C}(t) \oslash Adm_B(t)] .$$

Recalling the reading suggested in Section 2.1, we can re-write Def. 23 in a non-multivalued form as follows:

$$Good(B, G) \quad \text{iff} \quad \exists\, n > 0 \left[\forall\, t \in \mathcal{T}_n(S) \left(Adm_B(t) \supset G^{\downarrow C}(t) \right) \right] .$$

Thus, B is a good behavior for G if there is a number n such that any admissible execution of B that is longer than n satisfies G. The reason we disregard short trajectories is that they may not give the behavior enough time to accomplish G. For example, a one-step trajectory will obviously fail to satisfy the goal to reach a point distant from the agent, although it may be going in the right direction.

The subtle part of this definition lies in the interaction of the contextual conditions of B with the goal predicate. Recall that a trajectory is admissible for B if it is anchored and it follows the control schema D whenever in C (Def. 16). When a trajectory is out of the C context, any control is admissible, and the trajectory can vary arbitrarily. These trajectories are indifferent to the restricted goal $G^{\downarrow C}$. However, if some part of a trajectory enters the context, then this part must obey D; the definition above requires that this part also satisfy G. Hence, a behavior B is *Good* for a goal if, for every possible execution of B, either the context is never entered (or never entered long enough), or G is satisfied. [4] The following theorem formalizes this reading of *Good*.

Theorem 24 *Let $B = \langle C, D, O \rangle$ be a behavior schema, and G a goal. Then, $Good(B, G) \geq \alpha$ if, and only if, there is a positive integer N such that, for any trajectory t of length greater than N,*

$$Adm_B(t) \otimes C(t) \leq G(t) \oslash \alpha .$$

One consequence of our definition of *Good* is that we should not expect a behavior to work when its contextual conditions are unsatisfied. This is reasonable, as behaviors are always built to work under a given set of assumptions. Unfortunately, this implies that there is no way in general to guarantee that a

[4] Interestingly, this notion is similar to that of goal in Cohen and Levesque's theory of intention [CL90]: an agent keeps a goal until it is achieved, or until it is impossible. It has the same consequence: it is impossible to say whether a good behavior will ever actually achieve the goal.

behavior will reliably satisfy a goal in any environment: we can always imagine a malign environment that keeps an agent from staying in the context long enough, and hence from achieving the goal. What this means in practice is that, to be successful in dynamic environments, an agent needs to have behaviors that work in very general contexts, and/or the ability to change behavior if it goes out of context. We will come back on these issues in the next section.

Finally, we note two useful monotonicity properties of *Good*.

Theorem 25 *Let $B = \langle C, D, O\rangle$ and $B' = \langle C', D, O\rangle$ be two behavior schemas, and G, G' be two goals. Then, if $C' \subseteq C$ and $G \subseteq G'$*

(i) $Good(B, G') \geq Good(B, G)$
(ii) $Good(B', G) \geq Good(B, G)$

That is, goodness is preserved by relaxing the goal and by restricting the context. In the rest of this section, we will show how to build behaviors that are good for stronger goals and in larger context.

4.3 Compositionality

We know from the last section how to build complex behaviors by composing simpler ones. An important question is whether our composition operators preserve, in some way, the goodness of the component behaviors.

Theorem 26 (Conjunction of behaviors) *Let B_1 and B_2 be two behavior schemas, and G_1, G_2 be two goals. If $Good(B_1, G_1) \geq \alpha$ and $Good(B_2, G_2) \geq \beta$, then*

$$Good(\text{CONJ}[B_1; B_2],\, G_1 \cap G_2) \;\geq\; \min(\alpha, \beta)\,,$$

where $\cap$ is interpreted with respect to the min *t-norm.*

Theorem 26 tells us that we can build behaviors that address more goals simultaneously by conjoining behaviors that are good for the individual goals. As a typical example, a behavior for following a wall can be joined with a behavior for going fast (e.g., one that always prefers high speeds) to obtain a behavior to follow a wall quickly.

It is important to note that Theorem 26 only says that the conjoint behavior is good for the conjoint goal: it does not guarantee that either conjunction make sense. In practice, when conjoining behaviors, we should make sure that two consistency conditions are verified. First, that the goals are mutually consistent, that is, the value of $Sat(G_1 \cap G_2)$ is "reasonably" close to 1. For

instance, suppose G_1 and G_2 are the goals to reach two far apart locations. Then, $G_1 \cap G_2$ is empty, and promoting this goal will fail to promote each of the two conjuncts individually. The second condition is that the control strategies of the two behaviors should not interfere. Consider a room with two doors, and two behaviors, B_1 and B_2, for entering through the different doors. Each behavior is individually *Good* for its own goal, hence, the conjoined behavior CONJ$[B_1; B_2]$ is *Good* for the conjunction of the two goals, that is, entering the room. Unfortunately, there is no trajectory that is admissible for both B_1 and B_2, and the behavior cannot work properly. This situation can be detected by measuring the mutual consistency of the two behaviors, given by $Sat(\text{CONJ}[B_1; B_2]))$. Mutual consistency can be analyzed when designing (or planning) a combination; or it can be monitored during execution (see Sect. 5). The following theorem gives a necessary condition for consistency that can be tested at planning time: if B_1 falsifies the context of B_2, then B_1 and B_2 are inconsistent.

Theorem 27 *Let* $B_1 = \langle C_1, D_1, O_1 \rangle$ *and* $B_2 = \langle C_2, D_2, O_2 \rangle$ *be two behavior schemas. If* $Good(B_1, \text{ACHIEVE}[\neg C_2]) \geq \alpha$ *then* $Sat(\text{CONJ}[B_1; B_2]) \leq 1/2 \oslash \alpha$. *Moreover, if* $\alpha > 0$, *then* $Sat(\text{CONJ}[B_1; B_2]) < 1$.

The conjunction theorem allows us to build behaviors that satisfy stronger goals in a narrower context. We would like to also have a means of satisfying a goal in a wider context. Chaining of behaviors can provide this: if the first behavior in the combination achieves (according to the (9) above) the context of the second one, the two behaviors can cooperate to satisfy the goal of the second behavior in the union of the individual contexts.

Theorem 28 (Chaining of behaviors) *Let* B_1 *and* B_2 *be two behavior schemas, and* G *a goal. Then, if* $Good(B_1, \text{ACHIEVE}[C_2]) \geq \alpha$ *and* $Good(B_2, G) \geq \beta$, *then*

$$Good(\text{CHAIN}[B_1; B_2], G') \;\geq\; \min(\alpha, \beta)\,,$$

where $G' = G^{\downarrow C_2} \cup \text{SEQUENCE}[C_2, C_1]$, C_2 *is the context of* B_2, *and* $\cup$ *is interpreted with respect to the* max *t-norm.*

That is, given the goodness hypotheses, CHAIN$[B_1; B_2]$ is a good behavior for G (actually, G' — see below) under the more general context $C_2 \cup C_1$: intuitively, we use B_1 whenever in $C_1 \setminus C_2$ to reach the context C_2 of B_2; and then use B_2 to satisfy G. What this means in practice is that behavior chaining can be correctly used as a means for extending the effectiveness of the agent's motor skills beyond their (normally limited) context.

There is a technical caveat here. The chaining theorem says that CHAIN$[B_1; B_2]$ is *Good* for the disjunctive goal G'. The first component of G' is the restriction

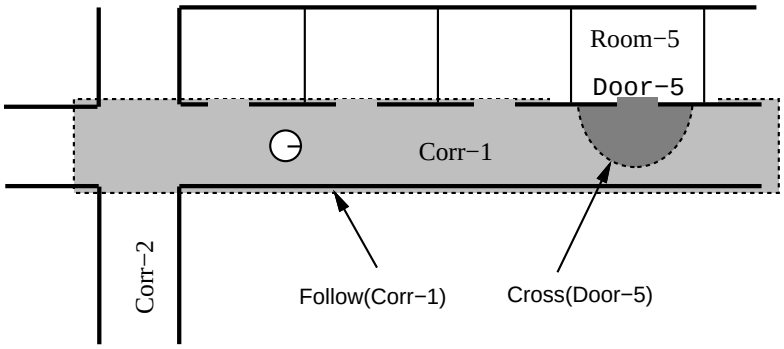


Fig. 11. A robot in an office environment. The dashed areas indicate the contexts of the Follow and the Cross behaviors.

of G to the context C_2: only the parts of the execution that are in C_2 are requested to satisfy G. As B_1 makes sure that the C_2 context is eventually entered, this corresponds to the goal of *eventually* satisfying G. For example, suppose a robot that has a behavior B_2 to maintain a fixed temperature in a room by switching a heater on and off; B_2 requires the robot to be in the room. If B_1 is a behavior for moving to the room from anywhere in the house, then CHAIN$[B_1; B_2]$ will be a good behavior for *eventually* fixing the temperature in the room if the robot is anywhere in the house.

Making sure that the behavior enters the context C_2 is not enough to guarantee that it will satisfy G: the behavior should also *remain* in C_2 long enough (the n parameter in the definition of *Good*). One way to guarantee this is to require that B_1 be good for the stronger goal ACHIEVE![C_2], but this may be unrealistic in some domains — it is unrealistic for many robot navigation behaviors. If we require simple achievement, there is no way in general to guarantee that trajectories will not go systematically out of C_2. If a trajectory stays in the context $C_1 \cup C_2$ of the chained behavior, however, there are only two possibilities: it keeps oscillating between C_1 and C_2; or it satisfies G. This is the meaning of the second component of G': admissible trajectories either satisfy G, or go back to C_1. Oscillatory behavior can be difficult to detect and to avoid. The following theorem gives a necessary condition for stability that can be tested at planning time: if each behavior promotes the context of the other one, their chaining will oscillate.

Theorem 29 *Let B_1, B_2 be two behavior schemas, and C_1, C_2 their contexts. If Good(B_1, ACHIEVE[C_2]) $\geq \alpha$ and Good(B_2, ACHIEVE[C_1]) $\geq \beta$, then*

$$Good(\text{CHAIN}[B_1; B_2], \text{SEQUENCE}[C_2, C_1]) \;\geq\; \min(\alpha, \beta)\,.$$

Example 30 *The following example illustrates chaining in practice. Consider the situation in Figure 11, where Flakey, sitting in corridor Corr-1, needs to reach room Room-5. Flakey has a control schema, called* Cross*, that results in crossing a door if applied when the robot is near to that door. That is, the*

following behavior schema is a good behavior (at some level, say 0.8) for the goal ACHIEVE[at(Room-5)]*:*[5]

$$B_1 = \langle\, \mathsf{near(Door\text{-}5)}, \mathsf{Cross(Door\text{-}5)}, \{\mathsf{Door\text{-}5}\} \,\rangle$$

where the Door-5 *object descriptor is kept anchored to Door-5. Unfortunately, the context of* B_1 *is not general enough to include the present situation. Fortunately, there is a way for Flakey to achieve this context. When applied inside a corridor, the* Follow *behavior schema seen in Example 15 results in Flakey eventually being* near *each object in that corridor. That is,*

$$B_2 = \langle\, \mathsf{at(Corr\text{-}1)}, \mathsf{Follow(Corr\text{-}1)}, \{\mathsf{Corr\text{-}1}\} \,\rangle$$

is a good behavior (say at the level 0.9) for achieving near(Door-5) *in our environment. The chaining theorem tells us that we can chain* B_1 *and* B_2 *to obtain a composite behavior,* CHAIN$[B_2; B_1]$*, that leads Flakey into Room-5 in the larger context of being anywhere along Corr-1. More precisely, and assuming that* B_1 *and* B_2 *do not oscillate, we have*

$$\mathit{Good}\,(\,\textsc{chain}[B_2; B_1],\ \textsc{achieve}[\mathsf{at(Room\text{-}5)}]\,)\ \geq\ \min(0.8, 0.9)\ =\ 0.8\,.$$

In the last example, we have reasoned by a form of goal-regression — although based on the less common notion of *extending the context* of applicability. We will see in the next section that the CONJ and CHAIN operators can be easily married to classical plan generation concepts and techniques.

4.4 Blending reactive and goal-oriented behavior

A central problem for autonomous agents operating in uncertain and dynamic environments is how to combine purposeful activity with sensitivity and responsiveness to (possibly unforeseen) events in the environment. For instance, an assembly robot should be prepared to promptly intervene if an assembly piece falls from the table; and a mobile robot should reliably avoid unforeseen or moving obstacles during goal-oriented navigation. Context-dependent blending of behaviors has proven in our experience to be a good method for combining reactivity and purpuseful action. Our practical methodology to write reactive goal-achieving behaviors has been as follows. We first write a

[5] We assume that the information about which basic behaviors are good for which goals, and to which degree, has been given by the designer of the behaviors (see next section).

simple goal-achieving behavior that assumes a somehow ideal context, for instance, that there are no obstacles. Then, we write a behavior that tryes to make (some of) these assumption hold when they do not — for instance, a behavior to go around convex obstacles. By chaining the latter behavior with the former one, we obtain a new behavior that achieves the goal under relaxed assumptions — in our example, under the assumption that there are only convex obstacles.

Example 31 *We reconsider the problem seen in Example 18 from the present perspective. The* Follow *behavior (like most behaviors) is written to be a good behavior for going down a corridor* provided that there are no physical impediments. *In environments with obstacles, it will easily go out of context. The* Keep-Off *behavior, on the other hand, is a good behavior for avoiding obstacles, that is, roughly said, for bringing about a situation where there are no more physical impediments. By chaining these behaviors, we can obtain a composite behavior that is Good for going down a corridor whether there are obstacles or not:*

CHAIN[⟨ facing(OG), Keep-Off(OG), {OG} ⟩ ;
⟨ at(Corr1), Follow(Corr1), {Corr1} ⟩] .

This behavior is equivalent to the one in Example 18 (by ignoring the Sense*). Note that, due to the blending mechanism, the desirability expressed by* Follow *is still considered, with decreased weight, during the obstacle avoidance maneuvers (see the plot in Figure 9). Thus, if the robot has a choice between different ways to avoid an obstacle, e.g., to pass it on the left or on the right, it will choose the avoidance strategy that is most compatible with the pursuit of the goal of* Follow.

Local combination of behaviors, of some form or another, is widely used in the robotic literature for mixing goal directedness and reactivity. One ubiquitous problem is the emergence of points of local equilibrium in the combination. Our approach gives us some control over these phenomena by allowing us to put disjoint contexts in the behaviors that may conflict, but local minima and oscillatory behaviors can still emerge in some cases. Global path-planning methods are better when two conditions hold: (a) the obstacles are complex, and (b) sensing gives an accurate picture of this complexity. Path-plannning techniques are not incompatible with our behavioral approach: an agent can be equipped with a path planning and following behavior, and combine it with other behaviors when needed. We have implemented such a behavior for Flakey using a method similar to Payton's [PRK90]. A gradient field is generated at intervals by a grid search from the goal position to the robot. At each point, the gradient gives the direction the robot should travel in the shortest path to the goal. A path-following control schema prefers the directions that are close

to those indicated by the field. This behavior is useful for coping with convex configurations of obstacles and escaping from local minima, provided that enough perceptual information has been gathered to show all the obstacles between the robot and its goal. It can also be blended with the more reactive Keep-Off behavior to take care of newly sensed obstacles.

5 Planning

An intelligent situated agent needs the ability to autonomously develop new strategies, or *plans*, for solving new tasks. This ability requires that the agent reason about its own motor skills, and about the relation between these skills and the achievement of goals under certain conditions. We have already introduced all the necessary ingredients in the previous sections: we have defined behavior schemas, contexts and goals; we have shown how to compose simpler behaviors into more complex ones; and we have studied the main relation between behaviors and goals: the *Good* predicate. Moreover, we have claimed that, while grounded in the physical level, these ingredients are at the right level of abstraction to be used by a reasoning process. In this section, we justify this claim by showing how we can automatically generate complex behaviors to satisfy given goals by using customary planning techniques. We refer to the automatically generated behaviors as *behavioral plans* to emphasize their double nature of planned activity and executable controls.

5.1 Reasoning about behavior

In order to reconcile our formalism with the more standard representations used in the planning tradition, we collect all the information about a basic behavior schema — i.e., one for which we have an implementation — in a data structure called a *behavior template*. For example, the following is the template relative to the Cross behavior used in Example 30.

```
      Template: CROSS
    Parameters: door(?d), connects(?p1,?d,?p2)
 PreCondition: (and at(?p1) near(?d) (not obstacle))
       Achieve: at(?p2)
ControlSchema: Cross(?d)
      Goodness: 0.9
```

What this template says is that if d is bound to a descriptor of a door and p1 and p2 to the descriptors of the two places connected by that door, then the

```
     Template: FOLLOW
   Parameters: corridor(?c), in(?c, ?p)
 PreCondition: (and at(?c) (not obstacle))
 RunCondition: anchored(?c)
      Achieve: near(?p)
ControlSchema: Follow(?c)
     Goodness: 0.9
```

```
     Template: SENSE
   Parameters: corridor(?c)
 PreCondition: at(?c)
      Achieve: anchored(?c)
ControlSchema: Sense(?c)
     Goodness: 0.7
```

Fig. 12. Templates for following and for anchoring a corridor.

behavior schema

$$B = \langle\, (\mathsf{at(p1)} \wedge \mathsf{near(d)} \wedge \neg\mathsf{obstacle}),\ \mathsf{Cross(d)},\ \{\mathsf{d},\mathsf{p1}\}\,\rangle$$

is such that

$$Good(B, \textsc{achieve}[\mathsf{at(p2)}]) \geq 0.9\,.$$

The value 0.9 measures the designer's confidence that the behavior will achieve its goal every time it is activated under the appropriate conditions. It is the responsibility of the basic behavior's designer to make sure that the knowledge encoded in a template correctly reflects the properties of the corresponding behavior schema. We can use templates to build behavioral plans by applying three operations: instantiation of templates, conjunction of behaviors, and chaining of behaviors. The theorems in Section 4.3 give us a lower bound of the goodness of the resulting plan given the goodness of the basic behaviors.

Example 32 *Recall the environment in Fig. 11, and consider the templates for the* Follow *and* Sense *behaviors shown in Fig. 12. Given the goal to reach Room-5, we instantiate the template for* Cross *shown above. As the context is not true in the present situation, we* CHAIN *an instance of the* Follow *template. The* RunCondition *in this template indicates a condition that should be maitained while the behavior is running, namely, that the corridor be anchored. Hence, we* CONJ Follow *with a concurrent instance of* Sense. *The resulting plan is given by*

$$\textsc{chain}[\,\textsc{conj}[\mathsf{Follow(Corr1)};\mathsf{Sense(Corr1)}];\mathsf{Cross(Door5)}\,]\,,$$

where we name behaviors after their control schema and object descriptor.

The last example illustrates the principles that guide the generation of a behavioral plan. Suppose an agent in state s who has a goal G. We know from Theorem 24 that, if a behavior schema $B = \langle C, D, O\rangle$ is good for a goal G, then any sufficiently long execution of B that is in the context C will satisfy G. Hence, the agent will be interested in building a behavior B such that: (1)

B is a good behavior for G; (2) there is an admissible execution t of B passing through s; and (3) the execution t lies entirely in the context C. Formally:

Definition 33 *Let $B = \langle C, D, O \rangle$ be a behavior schema, G a goal, and $s \in S$. The degree by which B is an* effective plan *for G in s is given by:*

$$\text{Plan}[B; G; s] = Good(B, G) \otimes \sup_{t: s \in t} \left(Adm_B(t) \otimes C(t)\right) .$$

It is difficult in general to evaluate the value of $\text{Plan}[B; G; s]$ prior to execution. We know how to compute goodness, but establishing the truth of the other two conditions is much harder. Theorem 27 gives us a condition for the consistency of B — hence for the existence of admissible trajectories — that can be verified on the templates: each behavior in a conjunction should not falsify the context of the other one. Unfortunately, this condition is only necessary. Computing the truth of $C(t)$ is problematic in general, and just impossible in dynamic environments. For example, the plan in Example 32 above assumes no obstacles. It is effective at a degree of (at least) 0.7 *given the environment in the picture,* but would go out of context (and then become ineffective) if an obstacle appears in front of the robot. In practice, the agent will have to monitor the conditions of Def. 33 during execution in order to detect situations where the plan becomes ineffective, and then modify it.

5.2 Pre-planned behavior

Our first experiment in the automatic generation of behavioral plans makes use a simple form of goal regression to generate a behavior that is good for the given goal, and whose context includes the starting state. We have implemented a small planner for Flakey based on a simple strategy: we start from an instance of a behavior that is good for the goal, and then enlarge its context by chaining behaviors that can achieve that context. We iterate the process until the context of the combined behavior covers the current state (or, in general, the set of situations for which we want to have a pre-computed response available.) If a behavior has a conjunctive precondition in its template, we use a correponding conjunction of behaviors to achieve it. As discussed above, care should be taken that the behaviors so conjoined be consistent. We currently delegate the detection of conflicts to a runtime monitoring process. [6]

Figure 11 shows a part of the office environment used in our experiments. Flakey has a topological map of this environment, annotated with some ap-

[6] We are also exploring the use of SIPE [Wil88] for building behavioral plans. SIPE uses "plan critics" to detect harmful interactions at planning time.

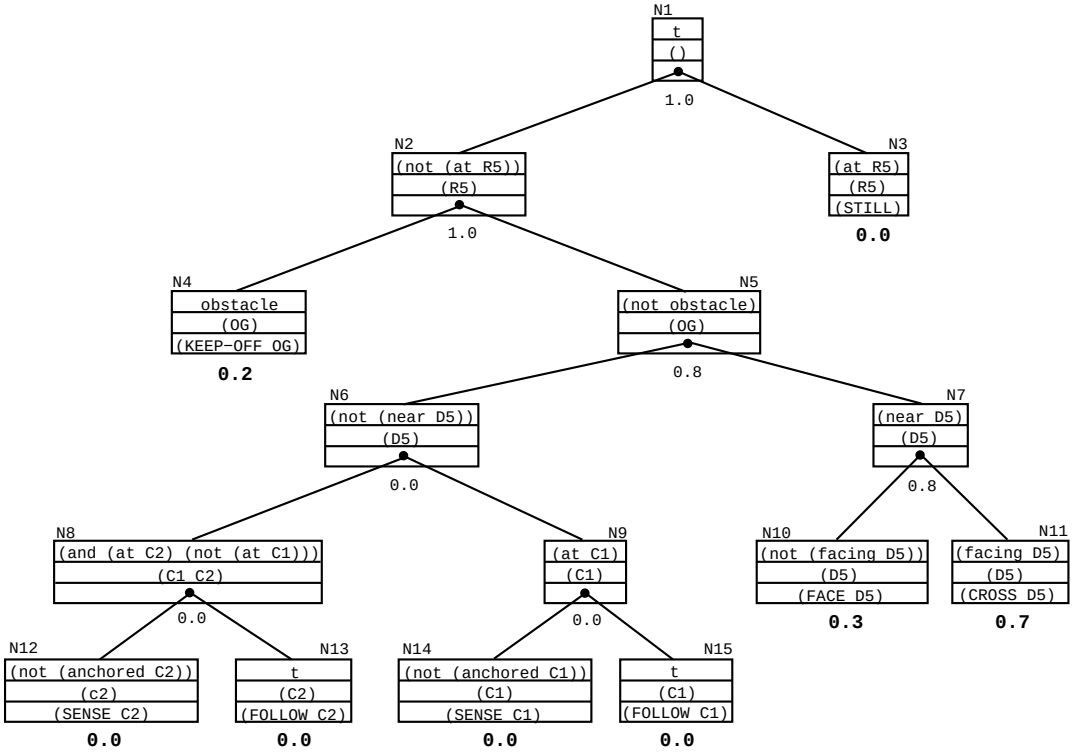


Fig. 13. A behavioral plan for reaching Room-5.

proximate metric data. Given this map and the goal to reach Room-5, Flakey's planner generated the behavioral plan shown in a graphical form in Figure 13. Each node in the plan represents a behavior schema, and lists the context, the names of the descriptors, and the control schema. For composite behaviors, the control schema is replaced by a pointer to the component behaviors. The overall tree represents a complex blending of the control schemas in the leaves.[7] The planner started by the Still behavior, and extended its context until it covered the starting situation. The Cross behavior implemented in Flakey is more restrictive than the one used above, and requires that the robot is (approximately) facing the door to cross. Correspondingly, the planner has chained a Face behavior to Cross. Note the use of the Keep-Off behavior to extend the applicability of the plan to situations where there are obstacles around.

Figure 14 shows an actual execution of this plan, along with a plot of the temporal evolution of the level of activation of the basic behaviors in the plan. Each behavior is activated when, and to the extent by which, its contextual

[7] Recall that, by using min for $\otimes$, every combination can be written in the canonical form (8), and then implemented by the context-dependent blending mechanism presented in Example 13. The truth value of each context is computed by propagating a unitary value down the tree, and ANDing (through min) it at each node with truth value of the context in that node. Figure 13 shows the values so computed in the situation g below.

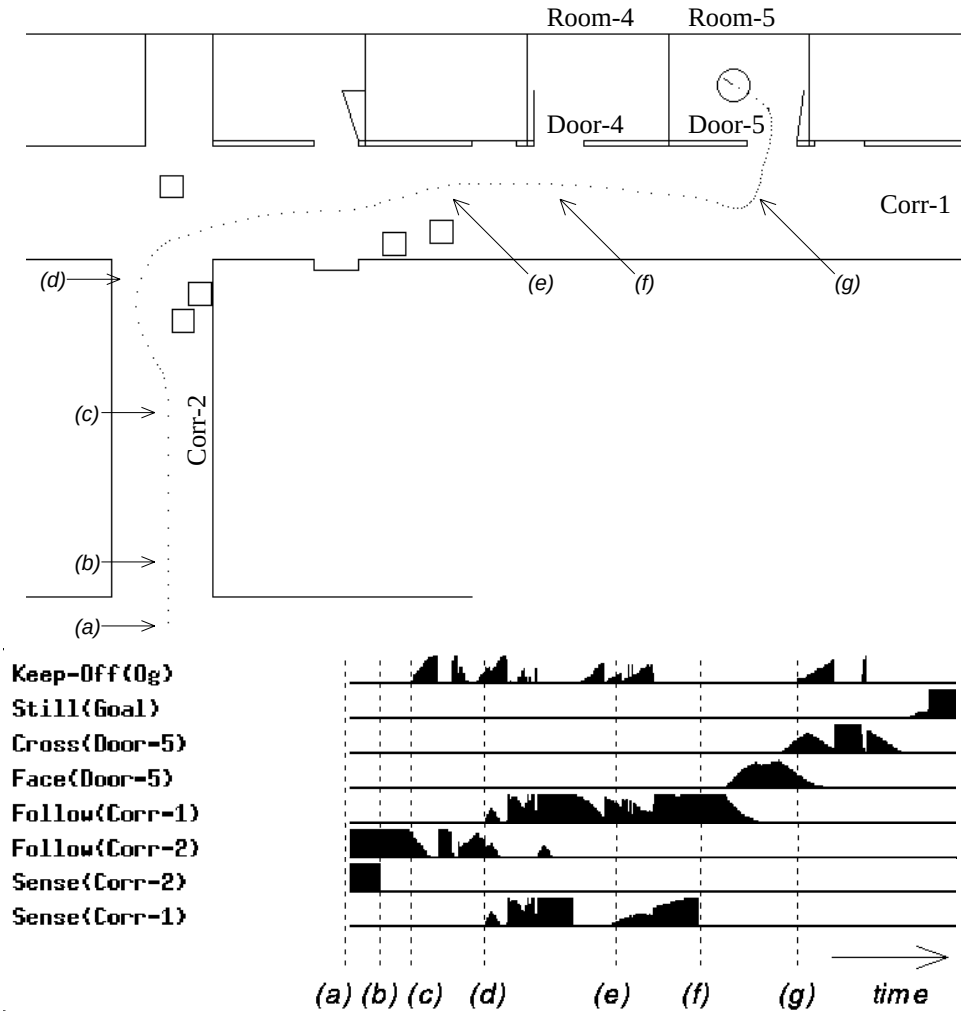


Fig. 14. Sample execution of the plan in Fig. 13.

conditions are verified. For behaviors that have been chained together, the activation of the first behavior is expected to produce the conditions for the activations of the second one, like in (d), and around (g). Note that sequencing is not determined by of an internal program counter, but it emerges from the interaction of the behaviors with the environment. Also notice that behaviors do not "terminate" in the usual sense: they are deactivated when their context become false (but may be reactivated later on). For example, when entering a new corridor (a,d), Flakey needs to anchor it by using the Sense behavior (Example 19); after traveling a few feet, the walls are anchored, Sense is deactivated, and Flakey can move more freely (b). If, however, a wall becomes occluded (e), Flakey may lose track of the corridor, and the context of Sense(Corr-1) gradually becomes true. Flakey then slows down until re-anchoring occurs (f). Clearly, no reasonable *a priori* ordering could have been established between the executions of the Follow and the Sense behaviors. Finally, note the interaction between purposeful behaviors and reactive obstacle avoidance after (c) and around (d), (e) and (g).

Generating a plan for future action inevitably brings about the problem of whether or not this plan will still be adequate when it is executed. A behavioral plan can be coupled with a *monitor* whose task is to detect when the plan fails to satisfy the three conditions in Def. 33, and correct it by introducing other behaviors to remedy the problem. Interestingly, we can compute an upper bound of the effectiveness of a plan using simple computations based on the current state.

Theorem 34 *Let $B = \langle C, D, O \rangle$ be a behavior schema and G a goal. Then, for any state $s \in S$,*

$$\mathrm{Plan}[B; G; s] \;\leq\; \mathit{Good}(B, G) \otimes \min[\inf_{a \in A} D(s, a), C(s)] \,.$$

There are three possible sources of a loss of effectiveness, corresponding to the three conditions in Def. 33. Condition (1), on the value of *Good*, can only become false if the goal is changed in some way. Condition (2) may fail if the plan contains an inconsistent conjunction of behaviors. When this happens, the agent gets into a stake state where the desirability function assigns a low value to all possible control actions, causing the value of $\inf_A D(s, a)$ to drop. Failure of condition (3) is the most common cause of failure of a plan: we have reached a situation for which the plan does not have any suggestion, so the value of $C(s)$ drops and all the behaviors have a low activation level. The solution is to extend the context to cover the current state. Often, this can be done by simply adding some new behaviors that achieve the context C of the original plan.

5.4 Run-time deliberation

Monitoring is not the only reason to push at least part of the deliberation process into execution. A generative planner like the one discussed above makes choices about which behavior to invoke for a given goal. Often these choices are better made at execution time, since there is more information available. For example, a robot encountering an obstacle in a hallway might choose to go around to the left or right, or gather more information before making a choice, depending on how close the obstacle is to one side, how much of the obstcle can be seen, and so on. Instead of generating a large, contingent plan to take care of every possible case, the planner could produce an initial partial plan, expecting to fill in behaviors only when it becomes necessary to satisfy an impending goal. One can think of runtime deliberation as tactical

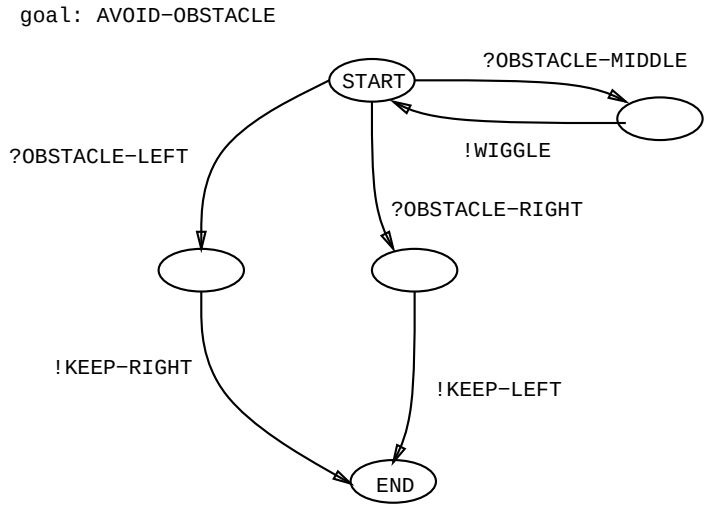


Fig. 15. A PRS intention schema.

planning: limited deliberation using rehearsed procedures for circumscribed tasks. Systems of this sort are called *reactive planners*. For our experiments, we adopted the Procedural Reasoning System (PRS) of Georgeff and Ingrand [GI89], whose main element is the *intention schema*, illustrated in Figure 15.

An intention schema is a finite-state machine whose arcs are conditions to test (**?P**) or to achieve (**!P**). The FS machine represents a limited strategy for achieving the goal of the schema. A sequence of behaviors is invoked by traversing the branches of the schema. A branch with a condition may be taken if the condition is satisfied. A branch with an achivement predicate causes the PRS interpreter to search for another schema whose goal matches the predicate, and invoke that schema. Behavior templates fit into this process as the lowest-level intention schemas, i.e., those that lead to action. The `Achieve` slot of the template can be matched to an achievement arc, and the behavior invoked by the interpreter to make the predicate true.

Example 35 *We implemented two strategies for making decisions during obstacle avoidance in corridors. In the first, we use a simple method for deciding which way to go around the obstacle. If the obstacle seems much further on one side, an immediate decision is made to go around on the other side. This decision may be the wrong one; as it moves, the robot will see more of the obstacle, and if it is blocked the opposite side is chosen. The second strategy is to acquire more information before deciding which way to move. On first contact with an obstacle, the robot "wiggles" to bring more of its sonars to bear on the obstacle, and then makes a decision. The decision on which strategy to use is made by a higher-level intention schema triggered by the presence of an obstacle (Fig. 15). Fig. 16 shows this schema at work while Flakey is patrolling a corridor up and down. In (a), the wrong decision is made initially, then corrected as more of the obstacle comes into view; (b) shows an extended*

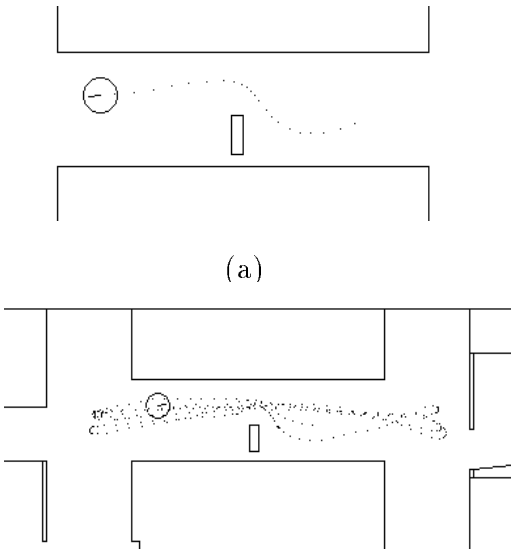


Fig. 16. Two runs of the obstacle-avoidance intention schema.

run.

Intention schemas are a way of instantiating (possibly complex) behaviors based on the current situation. A limited amount of deliberation can be performed to choose an appropriate schema when more than one will satisfy a given goal. We have used intention schemas to write strategies for different navigation tasks: moving around obstacles, moving into/out of rooms, deciding when to locate landmarks needed for self-localization, and even a complete plan execution and monitoring system. For the latter, we write intention schemas that examine the execution state of the plan and its currently-executing behaviors.

6 Related Work

The field of planning and control has burgeoned in the last few years, and many new ideas have emerged, especially in the area of reactive planning. The work we have presented owes much to previous work, and we have been influenced by methodologies and specific systems. In this section we give an overview of the points of contact, and draw attention to the distinct features of the multivalued logic approach to complex controllers.

Both the subsumption architecture of Brooks and his students [Bro82,Con90] and the situated automata of Rosenschein and Kaelbling [Ros87,KR90] are methodologies for producing embedded agents that perform complex tasks. In part we have borrowed from these in developing the complex behavior methodology, especially the subsumption idea of decomposing complex behavior into the composition of simple behaviors. In part we are in conflict with the spirit of these methodologies, in preferring explicit model-based perception and analogical representations of the world as part of embedding the controller.

The theory of situated automata is a formal methodology for constructing embedded agents by representing the environment of the agent, its task, and its capabilities. An automata is constructed to perform a task by considering these specifications [Kae88]. Situated automata theory is an abstraction away from the traditional planning approach in that it makes no commitment to an internal state that represents the world in an analogical fashion, i.e., that attempts to model surfaces, recognize objects, and so forth. The key observation of situated automata theory is that the state of the agent should contain just enough information to accomplish the specified tasks of the agent in its environment; and this information need not be in an analogical form. Situated automata theory is not incompatible with our approach; it just makes no commitment to any kind of internal architecture or representation. Ideally, we would like to be able to prove, using the techniques of situated automata, that any particular complex controller we design actually accomplishes its task in the intended environments. Further, we would like to be able to synthesize complex controllers that provably accomplish their goals. But the current state of situated automata theory is not developed enough to satisfy such an ambitious program. Its main practical success has been a suite of development tools: one of them, GAPPS, has been used to generate controllers for mobile robot navigation; we discuss it below.

The subsumption architecture has many points in common with situated automata theory, but without the formal emphasis of the latter. It is a task-oriented methodology for constructing agents. Each task is accomplished by a behavior, which integrates sensing, computation, and acting. The subsumption architecture is even stronger than situated automata theory in that it rejects the idea of a central, analogical representation of the environment. Each behavior is responsible for extracting needed information from the sensors, processing it in a task-dependent manner, and producing control actions. Behaviors are organized hierarchically, with the lowest level behaviors responsible for maintaining the viability of the agent, and the higher levels pursuing more purposeful goals. This vertical decomposition by behavior or task is contrasted with the horizontal decomposition of the traditional architecture, with

its expensive and nonreactive perceive/plan/execute cycle.

The subsumption architecture has been influential in the mobile robotic community, and its ideas have permeated most of the proposed architectures to some extent. We have incorporated the concept of vertical decomposition into the way in which behaviors interact with sensing and perception in our multivalued controllers: more reactive behaviors can access raw sensor readings, while more purposeful behaviors use more complex perceptual routines. In fact, the very notion of behaviors themselves is in the spirit of subsumption architecture, since each behavior is oriented towards accomplishing a particular goal. But our multivalued controllers differ in several important respects from the subsumption architecture. The most obvious one is a commitment to embedding and goal abstraction by means of a perceptual subsystem shared by all behaviors. Without such a subsystem, it is difficult to coordinate reactive and purposeful behavior in a general way, or to abstract the goals of behaviors so deliberation processes can make use of them, or to have behaviors whose purpose is to facilitate perceptual processes of recognition and anchoring.[8] The second main difference is in the way behaviors are combined together. Subsumptions architectures generally use by some form of a suppression mechanism. While these mechanisms may be adequate as a programming technique, it is difficult to prove formal properties about them, or to accomplish the kinds of sophisticated tradeoffs among goals that is available using multivalued logic.

6.2 Reactive planning architectures

There have been several proposals and implementations of hybrid architectures, ones that combine a low-level reactive control mechanism with one or more deliberative layers. The outline of a typical architecture is shown in Figure 17. The bottom layer is a controller, a bounded computation function from inputs and perhaps internal state to outputs. This layer usually implements some form of behavior-based control, in which the control function is composed from sub-functions that implement particular behaviors. The second layer initiates and monitors behaviors, taking care of temporal aspects of coordinating behaviors, such as deciding when they have completed their job, or are no longer contributing to an overall goal, or when environmental conditions have changed enough to warrant different behaviors. The tactical planner must complete its computations in a timely manner, although not as quickly as the control layer. In the top layer, long-term deliberative planning

[8] We note that there are some impressive results without such representation, e.g., Connell's can-retrieving robot [Con90] and Mataric's navigation experiments [Mat90].

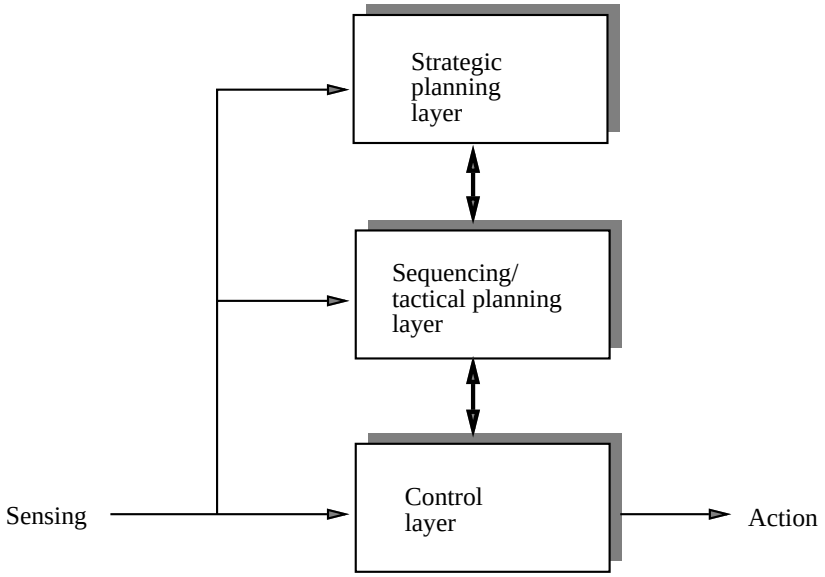


Fig. 17. A typical architecture for reactive planners (after Connell's SSS system).

takes place, with the results being passed down to the sequencing layer for execution. Generally, the planner is invoked and guided by conditions in the sequencing layer, e.g., a task failing or completing.

There are many different instantiations of this architecture, including SSS [Con92], ATLANTIS [Gat92], RAPs [Fir87,McD90], AURA [Ark90], and Pyton's reactive planners [PRK90]. Most of these concentrate on the interaction between the top two layers, developing sequencers and integrating them with planning technology. Our approach fits in the hybrid planning architecture, and we have concentrated on two important and not fully-developed aspects of this framework: the relation between the control and sequencing layers, and the principled composition of complex controllers. And we have developed the properties necessary to tie the control/sequencing level to the more abstract planning level.

6.3 Artificial potential-fields

Multivalued logics are one way to provide a trade off between concurrent goals. Another way is using techniques based on the so-called "artificial potential fields," first introduced by Khatib [Kha86] and now extensively used in the robotic domain [Lat91]. In the potential field approach, a goal is represented by a potential representing the desirability of each state from that goal's viewpoint. For example, the goal of avoiding obstacles is represented by a potential field having maximum value around the obstacles; and the goal of reaching a given location is represented by a field having minimum value at that location. At each point, the robot responds to a pseudo-force proportional

to the vector gradient of the field.

The major technical difference between our control schemas and potential-field methods is how they combine information. Potential fields are combined by linear superposition: one takes a weighted vector sum of the associated pseudo-forces. Each force is a summary of the preferences that produced that force (e.g., which direction is best to avoid an obstacle), and the combined force is a combination of the summaries. In contrast, when combining two control schemas one takes the t-norm of the two desirability functions, obtaining an assignment of utility values to *each* possible control. We have seen in Section 2.6 that these two forms of combination can produce different results. A second difference is that we express both goals and applicability conditions as formulae in a logical language. Complex goals and constraints can often be described more easily in a logical form than in the analytical form of a potential field function. Moreover, the ability to specify the context as a logical sentence makes integration with classical symbolic planning techniques easier. Potential-field approaches typically do not try to make contact with the symbolic planning methodologies.

A system based on potential fields and that has many points of contact with our approach is AuRA, developed by Arkin [Ark87,Ark90]. Arkin adopts Arbib's notion of motor schema, discussed in the Introduction. Motor schemas are similar to our concept of behavior, having the same basic components of control function, activation level, and perceptual parameters. However, AuRA implements control functions by using pseudo-forces instead of desirability functions. A related difference is that planning is integrated in a weaker form in AuRA. AuRA's planner is basically a path planner that generates a piecewise linear path to the goal. This plan is then passed to the execution layer, where motor schemas to follow this path are dynamically chosen and instantiated at execution time. In our approach, by contrast, the planner generates a complex composition of behaviors expected to orient the agent toward the achievement of a given goal. This approach is not limited in principle to navigation tasks.

6.4 Situation-action plans

A number of authors have claimed that plans to be executed by agents situated in the real world are better expressed in the form of "*situation* $\rightarrow$ *action*" rules. The triangle tables developed in early robot planning work [FN71] already incorporated the idea to represent, together with each action in the plan, the condition under which that action should be activated. More recently the idea of plans as sets of rules that specify reactions to situations has been clearly spelled out and extensively developed [Suc87,Sch87,Dru89,PRK90]. Our be-

havioral plans belong to this category.

Schoppers [Sch87] views planning as the task of partitioning the set of possible environmental situations according to the reaction that the agent should produce from the viewpoint of achieving the given goal. His *universal plans* specify a reaction to each possible situation that the agent can encounter, and can be thought of as controllers that provide an input-output mapping specific to the achievement of a given goal. Behavioral plans can be seen as a more control-oriented form of universal plans. The way we generate behavioral plans, by extending the context of an initial behavior to cover more and more situations, is similar to Schopper's notion of space partitioning. It is natural to ask how much we should enlarge this context. Two extreme solutions are to stop as soon as the context includes the current situation, as we did in Section 5.2; and to continue until we cover the entire state space, as proposed by Schoppers.

Most of the current agent architectures based on situation-action rules put these rules inside the control loop. This implies that the evaluation of the situation should be performed in bounded time, and that the actions should be elementary action steps. By considering this, Nilsson [Nil92] has recently extended the idea of triangle tables to develop a new formalism for plans, called *teleo-reactive trees*. A teleo-reactive tree encodes a set of situation-action rules, ordered by a subgoal relation: the activation of each rule in the tree is expected to eventually produce the conditions for the activation of its parent rule. Teleo-reactive trees resemble the behavioral plans that we have built in Section 5. In fact, a behavioral plan can be seen as a mathematically motivated generalization of a TR-tree where contexts can take intermediate degree of truth, desirability functions are multivalued, and we can have concurrent activation of behaviors.

A common feature of the approaches based on situation-action rules is that situation-action plans can be compiled into executable structures. Synthesis is a convenient property, because it automates the process of producing complex controllers. Another method for synthesizing complex controllers by compiling plans is GAPPS [Kae88]. Given a top-level goal, the GAPPS compiler reduces it using the goal-reduction rules into a set of condition-action rules for primitive control actions. Later development of GAPPS incorporated goal-regression compilation using operator descriptions [Kae90], which gives GAPPS a more abstract way of representing action, and enables it to do predictive planning. The two synthesis methods we explored for our multivalued controllers, goal-regression planning and run-time deliberation, are similar to GAPPS goal-regression and goal-reduction methods, respectively. Where the two systems differ is in the level of detail of complex action specification. We have concentrated on how control schemas combine to produce complex behavior. GAPPS control actions are simple effector commands, and conjoining

controls is only possible if they produce the same control action, or one of the control actions is unspecified. Although it is conceivable that GAPPS could be used to program multivalued logic control schemas, it would require the same development as we have presented here.

7 Conclusions

Intelligent action involves a continued interplay of local and global factors. Actions and sensing happen locally, in the here and now of the agent, but the goals they are aimed at, or the undesired consequences they can bring about, may lie far away in time and space. By planning, an agent tries to connect the actions it performs to its global goals and desires. But the result of planning, the *plan*, has to become physical activity. The work described in this paper focused on the relation between abstract goals of the plan and the physical actions that they induce in the agent.

Our approach to integrating planning and control has focussed on grounding the ingredients of planning in physical action, using the tools of multivalued logics. We started from the definition of basic types of movements, or *control schemas*, and of the way they can be combined or *blended* to form complex movements. Then, we have "lifted" control schemas to the level of abstract actions in an environment. Here, we have used two key notions: the notion of *embedding* in the environment, by anchoring the agent's internal state to external objects through perception, and the notion of *context*, or circumstances of execution. Finally, we have linked behaviors to *goals*, expressed as sets of satisfactory executions. The *good behaviors* for a goal are those that, when executed in an appropriate context, produce executions that satisfy the goal. And we have proven, under certain hypotheses, that composing behaviors creates a new behavior that is good for the composition of the corresponding goals. This result is the basis for automatic planning of complex behavior, and we have shown how traditional AI techniques for deliberation and means-ends analysis can be readily adapted to generate complex controllers for given goals.

One of the nice properties of the multivalued analysis of complex control is that it can be implemented using techniques from fuzzy control. We have illustrated this with examples from our mobile robot, Flakey. The ability of Flakey to navigate in unstructured real-world environments has been tested in innumerable runs in the corridors and offices of SRI during normal office activity. We have reported some of these experiments in Section 5. Flakey's performance has also been demonstrated in a few public events, including Flakey's second place at the first robotic competition of the AAAI, reported in [CHK+93], and a successor, Erratic, at the third competition [Kon95].

Our experience in the domain of mobile robot navigation has led us to formulate some methodological principles for achieving robust performance in dynamic environments:

- Behaviors should be computationally inexpensive, being simple functions on local conditions.
- To make them easy to write, debug, and compose (and possibly learn), behaviors should satisfy a single goal over a small range of environments.
- Complex behaviors to achieve multiple goals or operate over wider environmental conditions should be composed out of simpler ones.
- Since the environment will contain uncertainty, complex behaviors should be reactive, rather than relying on precomputed trajectories.

The work presented in this paper provides a formal framework for this compositional methodology, and enables the use of classical planning techniques to build complex behaviors. Our methodology is not a radical departure from the reactive planning methods now prevalent in the literature. Rather, it is a theoretical approach to two significant problems with these architectures: how to form complex movements, and how to link these up with more abstract deliberation processes.

Although the results obtained up to now are extremely promising, our study has uncovered a number of issues that need either a deeper formal analysis, or more experimentation, or both. Among the most urgent ones is the question of how the desirability functions used in the control schemas can be constructed in practice. We are currently studying techniques for synthesizing rules from abstract specifications of goals, and for improving these rules by learning methodologies. A second important issue that we only touched in this paper is the dynamic modification of planned behaviors. We are currently working on the further development of adequate indexes of performance, and on the use of these indexes to patch an existing plan. As for our formal development, two aspects that clearly need a deeper inspection are the notions of consistency and stability. We anticipate that studying these aspects will call for a richer representation of the state, including the dynamics of the environment. Finally, we feel that much more work is needed on the use of anchoring and object descriptors to relate perception and action.

Acknowledgments Nicolas Helft, David Israel and Daniela Musto contributed to the development of the ideas presented in this paper. Alessandro Saffiotti has been supported in part by a grant from the National Council of Research of Italy and in part by the Action de Recherches Concertees BELON funded by a grant from the Communaute Francaise de Belgique. Enrique Ruspini was supported by the U.S. Air Force Office of Scientific Research under Contract No. F49620–91–C–0060. Support for Kurt Konolige came partially

from ONR Contract No. N00014–89–C–0095. Additional support was provided by SRI International.

References

[AH85] M. Arbib and D. House. Depth and detours: an essay on visually guided behavior. Technical Report 85–20, COINS, University of Massachusetts, 1985.

[Ark87] R. C. Arkin. Motor schema based navigation for a mobile robot. In *Procs of the IEEE Int. Conf. on Robotics and Automation*, pages 264–271, 1987.

[Ark90] R. C. Arkin. Integrating behavioral, perceptual and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[Bel61] R. Bellman. *Adaptive control processes: a guided tour.* Princeton University Press, Princeton, NJ, 1961.

[Bro82] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, RA-2(1), 1982.

[BZ70] R. E. Bellman and L. A. Zadeh. Decision making in a fuzzy environment. *Management Science*, 17:141–164, 1970.

[CHK+93] C. Congdon, M. Huber, D. Kortenkamp, K. Konolige, K. Myers, E. H. Ruspini, and A. Saffiotti. CARMEL vs. Flakey: A comparison of two winners. *AI Magazine*, 14(1):49–57, Spring 1993.

[CL90] P. R. Cohen and H. J. Levesque. *Persistence, intention, and commitment.* MIT Press, Cambridge, MA, 1990.

[Con90] J. Connell. *Minimalist Mobile Robotics: A Colony-style Architecture for an Artificial Creature.* Academic Press, 1990.

[Con92] J. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 2719–2724, 1992.

[Dru89] M. Drummond. Situated control rules. In *Procs. of the 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 103–113, 1989.

[Fir87] J. R. Firby. An investigation into reactive planning in complex domains. In *Procs. of the AAAI Conf.*, 1987.

[FN71] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Gat92] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Procs. of the AAAI Conf.*, 1992.

[GI89] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Procs. of the AAAI Conf.*, pages 972–978, Detroit, MI, 1989.

[IPT91] D. Israel, J. Perry, and S. Tutiya. Actions and movements. In *Procs. of the Int. Joint Conf. on Artificial Intelligence*, Sydney, Australia, 1991.

[Kae88] L. P. Kaelbling. Goals as parallel program specifications. In *Procs. of the AAAI Conf.*, Minneapolis–St. Paul, MN, 1988.

[Kae90] L. P. Kaelbling. Compiling operator descriptions into reactive strategies using goal regression. Technical Report TR-90-10, Teleos Research, Palo Alto, CA, 1990.

[Kha86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[Kon95] K. Konolige. ERRATIC competes with the big boys. *AI Magazine*, Summer 1995.

[KR90] L. Kaelbling and S. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems*, 6:35–48, 1990.

[LA85] D. Lyons and M. Arbib. A task-level model of distributed computation for sensory-based control of complex robot systems. Technical Report 85–30, COINS, University of Massachusetts, 1985.

[Lat91] J.C. Latombe. *Robot Motion Planning.* Kluver Academic Publishers, Boston, MA, 1991.

[ŁT83] J. Łukasiewicz and A. Tarski. Untersuchungen über den Aussagenkalkül. *Comptes rendus de la Sociètè des Sciences et des Lettres de Varsovie (Cl. III)*, 23:157–168, 1983.

[Mat90] M. Mataric. A distributed model for mobile robot environment learning and navigation. Technical Report 1228, MIT AI Laboratory, 1990.

[McD90] D. McDermott. Planning reactive behavior: A progress report. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.

[MS90] C. Malcom and T. Smithers. *Symbol grounding via a hybrid architecture in an autonomous assembly system*, pages 123–144. MIT Press, Cambridge, MA, 1990.

[Nil92] N. J. Nilsson. Toward agent program with circuit semantics. Technical Report STAN–CS–92–1412, Stanford University, Computer Science Dept., Stanford, CA, 1992.

[Ove84] K. Overton. *The Acquisition, Processing, and Use of Tactile Sensor Data in Robot Control.* PhD thesis, University of Massachusetts at Amherst, 1984.

[PRK90] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey. Plan guided reaction. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(6), 1990.

[Res67] N. Rescher. Semantic foundations for the logic of preference. In N. Rescher, editor, *The Logic of Decision and Action.* Pittsburgh, 1967.

[Res69] N. Rescher. *Many Valued Logic.* McGraw-Hill, New York, 1969.

[Ros87] S. J. Rosenschein. The synthesis of digital machines with provable epistemic properties. Technical Note 412, SRI Artificial Intelligence Center, Menlo Park, California, 1987.

[Rus90] E. H. Ruspini. Fuzzy logic in the Flakey robot. In *Procs. of the Int. Conf. on Fuzzy Logic and Neural Networks (IIZUKA)*, pages 767–770, Japan, 1990.

[Rus91a] E. H. Ruspini. On the semantics of fuzzy logic. *Int. J. of Approximate Reasoning*, 5:45–88, 1991.

[Rus91b] E. H. Ruspini. Truth as utility: A conceptual synthesis. In *Procs. of the 7th Conf. on Uncertainty in Artificial Intelligence*, Los Angeles, CA, 1991.

[Saf94] A. Saffiotti. Pick-up what? In C. Bäckström and E. Sandewall, editors, *Current Trends in AI Planning*, pages 166–177. IOS Press, Amsterdam, Nederlands, 1994.

[Sch87] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Procs. of the Int. Joint Conf. on Artificial Intelligence*, pages 1039–1046, 1987.

[Seg85] K. Segerberg. Routines. *Synthese*, 65:185–210, 1985.

[SKR93] A. Saffiotti, K. Konolige, and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. Technical Report 533, SRI Artificial Intelligence Center, Menlo Park, California, 1993.

[SRK93a] A. Saffiotti, E. H. Ruspini, and K. Konolige. A fuzzy controller for Flakey, an autonomous mobile robot. Technical Report 529, SRI Artificial Intelligence Center, Menlo Park, California, 1993.

[SRK93b] A. Saffiotti, E. H. Ruspini, and K. Konolige. Integrating reactivity and goal-directedness in a fuzzy controller. In *Procs. of the 2nd Fuzzy-IEEE Conference*, pages 134–139, San Francisco, CA, 1993.

[SS83] B. Schweizer and A. Sklar. *Probabilistic metric spaces.* North-Holland, Amsterdam, NL, 1983.

[Suc87] Lucy Suchman. *Plans and situated actions: the problem of human machine communication.* Cambridge University Press, Cambridge, MA, 1987.

[Val85] L. Valverde. On the structure of F-indistinguishability operators. *Fuzzy sets and systems*, 1985.

[Web83] S. Weber. A general concept of fuzzy connectives, negations and implications based on t-norms and t-conorms. *Fuzzy sets and systems*, 11:115–134, 1983.

[Wil88] D. E. Wilkins. *Practical Planning.* Morgan Kaufmann, San Mateo, CA, 1988.

[YP92] J. Yen and N. Pfluger. A fuzzy logic based robot navigation system. In *Procs. of the AAAI Fall Symposium on Mobile Robot Navigation*, pages 195–199, Boston, MA, 1992.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

8 Appendix: proofs

We prove here the main theorems in this paper. The proof of the other results can be found in the technical report version [SKR93]. The following properties of t-norms will be used in the subsequent proofs.

Lemma 36 *Let $\otimes$ be any a continuous T-norm with quasi-inverse $\oslash$. Then*

(i) $\sup_S[f(w) \otimes g(w)] \leq \sup_S f(w) \otimes \sup_S g(w)$.
(ii) $\inf_S[f(w) \otimes g(w)] \geq \inf_S f(w) \otimes \inf_S g(w)$.
(iii) $x \oslash y \geq z$ if and only if $x \geq z \otimes y$.
(iv) $(x \oslash z) \otimes y \leq (x \otimes y) \oslash z$.
(v) $(x \oslash y) \oslash z = x \oslash (y \otimes z)$.
(vi) $(x \otimes y) \oslash (z \otimes w) \geq (x \oslash z) \otimes (y \oslash w)$.

Proof of Theorem 10

Looking first at the relation between $Next_{D\downarrow C}$ and $Next_D$, we may notice that, for any two states s, s' in S, it is

$$\begin{aligned} Next_{D\downarrow C}(s,s') &= \sup_{a\in A}\,[(D(s,a) \oslash C(s)) \otimes M(s,a,s')] \\ &\leq \sup_{a\in A}\,[(D(s,a) \otimes M(s,a,s')) \oslash C(s)] \\ &= Next_D(s,s') \oslash C(s)\,. \end{aligned}$$

Let now $t = (s_0, s_1, \ldots, s_k)$ be any trajectory, and $t' = (s_l, s_{l+1}, \ldots, s_m)$, with $0 \leq l < m \leq k$, any subtrajectory of t. Then we can see that

$$\begin{aligned} Traj_{D\downarrow C}(t) &\leq \inf_{0\leq i<k}\,[Next_D(s_i,s_{i+1}) \oslash C(s_i)] \\ &\leq \inf_{l\leq i<m}\,[Next_D(s_i,s_{i+1}) \oslash C(s_i)] \\ &\leq \inf_{l\leq i<m} Next_D(s_i,s_{i+1}) \oslash \inf_{l\leq j\leq m} C(s_j) \\ &= Traj_D(t') \oslash C(t')\,. \end{aligned}$$

As the above inequality holds no matter what subtrajectory t', we have proved the theorem. □

Proof of Theorem 24

By definition, $Good(B, G) \geq \alpha$ is true iff there is a positive number N such that, for any trajectory $t \in \mathcal{T}_N(S)$, it is

$$G^{\downarrow C}(t) \oslash Adm_B(t) \ \geq \ \alpha \, .$$

From this, and by applying Lemma 36 (iii), we have

$$\begin{aligned} Adm_B(t) \otimes C(t) &\leq (G^{\downarrow C}(t) \oslash \alpha) \otimes C(t) \\ &\leq (G^{\downarrow C}(t) \otimes C(t)) \oslash \alpha \\ &\leq [(G(t) \oslash C(t)) \otimes C(t)] \oslash \alpha \\ &\leq G(t) \oslash \alpha \, . \quad \Box \end{aligned}$$

Proof of Theorem 26

We let $B = \textsc{conj}[B_1; B_2]$. For any trajectory t, we have

$$\begin{aligned} Adm_B(t) \otimes C_2(t) &= Traj_{(D_1 \cap D_2)^{\downarrow C_1 \cap C_2}}(t) \otimes Anch_{O_1 \cup O_2} \otimes C_2(t) \\ &\leq Traj_{D_1^{\downarrow C_1 \cap C_2}}(t) \otimes C_2(t) \otimes Anch_{O_1} \\ &\leq [Traj_{D_1^{\downarrow C_1}}(t) \oslash C_2(t)] \otimes C_2(t) \otimes Anch_{O_1} \\ &\leq Traj_{D_1^{\downarrow C_1}}(t) \otimes Anch_{O_1} \\ &= Adm_{B_1}(t) \, , \end{aligned}$$

where we have used Theorem 10 for the third passage. From $Good(B_1, G_1) \geq \alpha$ by wirtue of Theorem 24, there exists N_1 such that $Adm_{B_1}(t) \otimes C_1(t) \leq G_1(t) \oslash \alpha$. for any trajectory t of length greater than N_1. A similar property holds for B_2 and G_2 for, say, N_2. Let $N = \max(N_1, N_2)$. Then, for any $t \in \mathcal{T}_N(S)$,

$$\begin{aligned} Adm_B(t) \otimes C_1(t) \otimes C_2(t) &\leq \min(G_1(t) \oslash \alpha, \, G_2(t) \oslash \beta) \\ &\leq \min(G_1(t), G_2(t)) \oslash \min(\alpha, \beta) \, , \end{aligned}$$

and the thesis follows from Theorem 24. $\Box$

Proof of Theorem 28

Let $B = \text{CHAIN}[B_1; B_2]$, and let $C = C_1 \cup C_2$. We first consider the two goodness hypotheses. By Theorem 24, the hypothesis on B_1 means that

$$Adm_{B_1}(t) \otimes C_1(t) \;\leq\; \sup_{s \in t} C_2(s) \oslash \alpha$$

for any trajectory of length greater than some suitable integer, call it N_1. For each such trajectory t, we then have

$$Adm_B(t) \otimes C(t) \;\leq\; \sup_{s \in t} C_2(s) \oslash \alpha\,. \tag{14}$$

On the other hand, the hypothesis on B_2 translates to

$$Adm_{B_2}(t) \otimes C_2(t) \;\leq\; G(t) \oslash \beta$$

for any trajectory t of length greater than a second suitable integer, call it N_2. Hence we have, for any such trajectory,

$$\begin{aligned} Adm_B(t) \otimes C_2(t) &\leq Traj_{D_2^{\downarrow C_2}}(t) \otimes Anch_{O_2}(t) \otimes C_2(t) \\ &= Adm_{B_2}(t) \otimes C_2(t) \\ &\leq G(t) \oslash \beta\,. \end{aligned}$$

By monotonicity of goal restriction, the last inequality implies that

$$Adm_B(t) \otimes C_2(t) \leq G^{\downarrow C_2}(t) \oslash \beta\,. \tag{15}$$

We then turn attention to our thesis. By Theorem 24, we need to prove that there is a positive integer N such that, for any trajectory t longer than N,

$$Adm_B(t) \otimes C(t) \leq G'(t) \oslash \min(\alpha, \beta)\,,$$

or, equivalently,

$$[G^{\downarrow C_2} \cup \text{SEQUENCE}[C_2, C_1]](t) \geq Adm_B(t) \otimes C(t) \otimes \min(\alpha, \beta)\,. \tag{16}$$

Let $N = N_1 + \max(N_1, N_2)$, and let $t = (s_0, s_1, \ldots, s_n)$ be any trajectory of length greater than N. From (14), there must be some state $s \in t$ such that

$$C_2(s) \geq \alpha \otimes Adm_B(t) \otimes C(t)\,.$$

Let k be the smallest index of such a state, and let $t' = (s_k, s_{k+1}, \ldots, s_n)$ be the part of t beyond s_k. By the goodness hypothesis on B_1, $k \leq N_1$, and so t' must have length at least $\max(N_1, N_2)$. We show that t' satisfies the goal G'. For suppose it does not. Then (16) must be false of t', that is, we must have both

$$G^{\downarrow C_2}(t') < Adm_B(t') \otimes C(t') \otimes \min(\alpha, \beta) \tag{17}$$

and

$$\textsc{sequence}[C_2, C_1](t') < Adm_B(t') \otimes C(t') \otimes \min(\alpha, \beta)\,. \tag{18}$$

Consider (18). From the definition of SEQUENCE and from (14) we have

$$\begin{aligned}
\textsc{sequence}[C_2, C_1](t') &= \sup_{k \leq i < n} \sup_{i < j \leq n} [C_2(s_i) \otimes C_1(s_j)] \\
&\geq \sup_{k \leq i < n} [C_2(s_i) \otimes \inf_{k \leq j < n} C_1(s_j)] \\
&\geq \sup_{k \leq i < n} C_2(s_i) \otimes C_1(t') \\
&\geq Adm_B(t') \otimes C(t') \otimes C_1(t') \otimes \alpha\,.
\end{aligned}$$

From this, and by applying (18), we have

$$Adm_B(t') \otimes C(t') \otimes C_1(t') \otimes \alpha < Adm_B(t') \otimes C(t') \otimes \beta$$

from which we conclude

$$C_1(t') < \beta \oslash \alpha\,.$$

By putting this in (17), we obtain

$$\begin{aligned}
G^{\downarrow C_2}(t') &< Adm_B(t') \otimes C(t') \otimes \min(\alpha, \beta) \\
&\leq Adm_B(t') \otimes C_2(t') \otimes C_1(t') \otimes \alpha \\
&< Adm_B(t') \otimes C_2(t') \otimes (\beta \oslash \alpha) \otimes \alpha \\
&\leq Adm_B(t') \otimes C_2(t') \otimes \beta\,.
\end{aligned}$$

But this contradicts the goodness hypothesis on B_2 (15). Hence one of (17) and (18) must be rejected, and (16) must hold of t', thus proving the theorem. □