SRI International

A MULTIVALUED LOGIC APPROACH TO INTEGRATING PLANNING AND CONTROL

Technical Note No. 533

June 1993

By: Alessandro Saffiotti†, International Fellow
Kurt G. Konolige, Sr. Computer Scientist
Enrique H. Ruspini, Sr. Computer Scientist
Artificial Intelligence Center
Computing and Engineering Sciences Division

† Currently at: IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

Alessandro Saffiotti was supported by a grant from the National Council of Research of Italy. Research performed by Enrique Ruspini, leading to the conceptual structures used in the autonomous mobile vehicle controller, was supported by the U.S. Air Force Office of Scientific Research under Contract No. F49620-91-C-0060. Support for Kurt Konolige was partially provided for by the Office of Naval Research under Contract No. N00014-89-C-0095., and additional support was provided by SRI International.

The views, opinions and/or conclusions contained in this note are those of the author and should not be interpreted as representative of the official positions, decisions, or policies, either expressed or implied, of the U.S. Air Force Office of Scientific Research, the Office of Naval Research, or the United States Government.

333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 1993		**00-06-1993 to 00-06-1993**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A Multivalued Logic Approach to Integrating Planning and Control	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **SRI International,333 Ravenswood Avenue,Menlo Park,CA,94025**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**96**	

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 *control structures*, which link physical movements to abstract action descriptions. Control structures induce behaviors of an agent, expressed as trajectories of control actions in an environment, and goals can be defined as predicates on these trajectories. By using the operations of multivalued logic, goals and behaviors can be combined to produce conjoint goals and complex controls. The ability of multivalued logic to represent intermediate degrees of goal satisfaction allows us to formulate trade-offs between competing goals. A composition theorem relates complex controls to conjoint goals, and provides the key to using standard deliberation procedures to generate complex controllers. We describe experiments in both planning and run-time deliberation on a mobile robot platform, Flakey.

Contents

1 Introduction

Autonomous operation of an agent embedded in a real environment (e.g., a mobile robot) poses a series of problems. Knowledge of the environment is partial and approximate; sensing is noisy; the dynamics of the environment can be only partially predicted; and an agent's hardware execution is not completely reliable. Still, the agent must take decisions and execute actions at the time scale of the environment. Classical planning approaches have been criticized for not being able to cope adequately with this situation, and a number of reactive approaches to robot control have been proposed (e.g., [Firby, 1987; Kaelbling, 1987; Gat, 1991; Yen and Pfluger, 1992]). Reactivity provides immediate response to unpredicted environmental situations by giving up the idea of reasoning about future consequences of actions. However, reasoning about future consequences (sometimes called "strategic planning") is still needed in order to intelligently solve complex tasks (e.g., by deciding not to carry an oil lantern downstairs to look for a gas leak [Firby, 1987].)

One solution to the dual need for strategic 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. 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 bridging the gap between abstractly specified goals and physical movements, and trading off between multiple goals.

In this paper, we adopt the complex controller methodology, and develop a particular form of controller that has distinct advantages:

- The controller is based on multivalued logic. The atomic units of the controller, called *control schemas*, induce a preference among control actions. Multiple control schemas can be composed mathematically as a means of trading off between their goals.
- 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.
- The main result linking goals and control schemas is that conjoined goals can be achieved by composing their respective control schemas. Thus, standard delibera-

tion procedures, such as goal-regression planning, can be used to generate complex controllers.

1.1 Multivalued Logic and Control Schemas

Multivalued logic provides a formal framework for understanding the action of the controller [Ruspini, 1990; Ruspini, 1991a]. The importance of using multivalued logic arises from the nature of the control problem, which involves complex trade-offs between competing goals. Our approach is to decompose the problem into small units of control, called control schemas, which are an implementation of one specific motor skill resulting in a certain type of physical movement. Control schemas define the agent's basic movement capabilities: for a human they include making a step or lifting the left hand; for a wheeled robot, they may include rolling forward or pointing the camera to the right. Control schemas should be simple to write and debug, because they have a limited focus. In normal activity, many control schemas are combined to produce complex movements: biking to the end of Alma Street involves an intricate combination of movements. This is also true for most popular approaches to building artificial agents — advanced execution skills are normally implemented by combining more basic motor controls. It is important that the schemas not impose strict constraints on control actions, or else two schemas would always conflict, and no combination would be consistent. Thus, we implement control schemas using multivalued functions from internal states to control actions, so that they induce a preference over action. In this way two control schemas can be combined when they overlap in the right way, that is, the areas where one schema has a strong preference are indifferent to each other. For example, a control schema for going down a corridor may not care whether it is exactly centered in the corridor, as long as it is not too close to the walls. Another control schema might want to stay to the left because it is avoiding watercoolers along the right-hand wall. The control actions from these two schemas can be blended, using the mechanisms of multivalued logic, to produce the behavior of going down the corridor to the left of center. Because of soft constraints on control actions, the blending of control schemas produces smooth motion that interpolates their respective behaviors.

Another advantage of using multivalued logic is that complex contextual conditions are easy to specify, because the logic provides operators for combining predicates. Suppose an active vision system is tracking an object, and it wants to keep tracking the object as long as it is not too far away, and no other interesting object comes closer. A contextual predication for the tracking behavior would be:

$$P(a) = near(a) \wedge \neg \exists x.closer(x, a) ,$$

where a is the tracked object. As long as $P(a)$ is true, the tracking behavior is active. Since we use a multivalued logic, the predication can take on degrees of truth: as the object moves further away, $P(a)$ becomes less true, and the activation of the tracking behavior becomes less important and subject to preemption by other behaviors.

1.2 Specifying and Combining Goals

Combining control schemas in the right way is in general a difficult problem, and may require informed deliberation: in the biking example, I need to do some amount of planning to decide to begin to slow down long before the intersection with Willow Road because my brakes are bad. One of the hardest problems in building complex controllers is linking them to higher-level planning processes of this sort, because planners use a more abstract view of action than the physical movements output by control schemas. Planners generally neglect issues of execution and physical movement, many of which are substantial and can severely limit the applicability of the planner. First, *actions are not procedures* to be rigidly executed: the same action can usually be instantiated by many movement executions, depending on the environmental and subjective conditions — for example, my hand can follow an infinite number of trajectories when performing the action of grasping a book. Generally, actions are defined abstractly in a planner by their goals: *pickup*($?A$) is an operator whose effect is to grasp a block. This operator description does not give even a hint as to what physical movements to perform. Second, *actions may fail*, that is, they may not bring about their expected effect. Consequently, a rigid ordering of actions that results in a precise course of events cannot in general be reliably anticipated — later actions may become vain if previous ones fail. Finally, *the objects of the actions are outside the agent* — they are not internal data structures that can be manipulated by a procedure. In order to perform an action, the agent needs to *anchor* its internal descriptions of the objects of an action to physical objects in the environment. These problems, and the first two in particular, have been the subject of intensive study in recent years (e.g., [Brooks, 1987; Schoppers, 1987; Kaelbling, 1988; McDermott, 1990b; Gat, 1991; Nilsson, 1992; Saffiotti, 1993]).

Our approach to linking physical movements to abstract action descriptions follows a path that goes from bodily movements, to the execution of movements under specific circumstances, or *behavior*, to the goal of a behavior. This way to connect movements, actions, and achievements has been inspired by [Israel *et al.*, 1991]. In our case, the main ingredients are control schemas as a way to describe bodily movements, goals given in terms of properties of executions, and contextual circumstances in which execution takes place.

The mathematics of multivalued logic provides a natural way to connect these ingredi-

ents. Control schemas induce trajectories of control actions of the agent, corresponding to all possible executions; contextual predicates select some of these trajectories as relevant; and goals can be defined as predicates on the trajectories. For example, suppose $B(a)$ is a control schema for picking up a block a. We can show that it accomplishes this goal under certain 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.

Now for the essential part of our development. We can show that individual control schemas accomplish a goal, and we can compose control schemas to form complex controls. What we would like to prove is that the composition of control schemas accomplishes a corresponding composition of their goals. If this were true, it would be the key to a compositional approach to behavior. Fortunately, under certain consistency conditions, we are able to prove some general theorems that relate composed control schemas to composed goals. These theorems have immediate consequences for the concurrent execution of control schemas and for their automatic generation.

Suppose we have a control schema that satisfies a given goal under certain conditions. There are at least two reasons why we may want to compose this schema with other control schemas: to have a control that satisfies a more specific goal, or to have a control that can be used under more general circumstances.

As an example of the first case, consider two control schemas B_1 and B_2, that achieve the goals to follow a corridor, and to move fast, respectively. We can compose these schemas into a control that achieves the stronger goal to follow a corridor fast. A *conjunctive composition* theorem shows that if the two goals are mutually consistent, that is, there is some trajectory of B_1 and B_2 that satisfies both of them, then composing B_1 and B_2 makes a complex control that satisfies their conjunction. In our example, to the extent that moving fast does not interfere with the navigation necessary to follow the corridor, the composite control will issue commands that follow the corridor at a high rate of speed.

An interesting example where we need to extend the applicability of a control schema to more general circumstances is the blending of reactive and purposeful movement. A control schema to reach a goal point can be easily developed under the condition that there are no obstacles to the goal. This context is too restrictive to be useful: when the robot senses an obstacle, this schema is no longer applicable. Another control schema can be written that, in the context of a sensed obstacle, moves the robot around the obstacle. Composing these schemas yields a complex control that is competent to reach a goal point in the presence of obstacles. When the robot is close to an obstacle, the avoidance behavior is primary; when

there are no obstacles, the goal-seeking behavior takes over. The key observation here is that the second control schema creates the conditions for applying the first one. A *chaining composition* theorem guarantees that the combined control achieves the goal of the first control schema in the disjunction of the two contexts.[1]

The "good behavior" of a composed control is based on some prerequisites: for conjunction, that the two goals be compatible, and for chaining, that the first control schema make the context of the second one true. The use of multivalued logic enables us to capture situations where these prerequisites are only partially true. For instance, two goals that are only somewhat compatible can be traded off when conjoined. The resultant complex control will satisfy each of the goals to some extent. The composition theorems provide a quantification of the *goodness* of a complex control as a function of the truth of the prerequisites for composition, and of the *goodness* of the component controls.

One of the consequences of the composition theorems is that standard deliberation procedures, which take operator descriptions and goals as input, can be used to generate complex controllers. We have experimented with two examples of these procedures, a simple goal-regression planner, and a goal-reduction execution process.

1.3 Perception and Anchoring

Finally, there is the problem of connecting the internal state used by control schemas to the environment. Again, we are faced with a problem of abstraction. Typically, controllers operate on sensor input, with interpretation of the sensors only implicit in the equations of the controller. Yet the goals of control schemas are expressed in terms of objects in the world, for example, picking up a particular block. Our approach is to use model-based perception to produce *artifacts*, internal representations of objects, which are then used as input to control schemas. This "lifts" the effects of the controller to the level of abstraction required to provably achieve a goal. Model-based perception has been standard methodology for traditional planning systems, which acquire a snapshot of the environment, interpret it, form a plan, and then execute it by issuing commands to a controller. The major complaint of this approach is that it is not reactive; the controller uses the information passed to it from the planner, e.g., the position of a doorway to traverse, and does not update it during movement. Our use of artifacts allows the controller to keep track of relevant objects during execution, by polling perceptual routines to keep the artifact anchored to its real-world correspondent.

[1]There are some caveats, involving how much of the trajectory lies in each context. In general there is no way to predict whether the composed control causes infinite oscillation between the two behaviors.

Of course, there is still the problem of assuring that the perceptual system anchors the internal representation in the right way, and there is the additional problem of maintaining reactivity, because perception can be computationally expensive. A complete solution to these problems is obviously difficult and not in the scope of this work. Still, we have developed a framework for integrating a perceptual architecture that, given the limitations of our current perceptual apparatus, is a sufficient solution. The key construct is the Local Perceptual Space (LPS), a representation of local space that integrates low-level sensor data and artifacts. Control schemas can use input from the LPS at an appropriate level of interpretation, that is, reactive behaviors use direct sensor data, while more purposeful movement behaviors use artifacts that are constructed and tracked by more complex perceptual routines. Finally, some behaviors can moderate the interaction of perceptual routines, by producing movements that favor information gathering and interpretation necessary for other behaviors.

1.4 Experiments with Flakey

We have tested the complex control methodology extensively on indoor robot navigation problems, using a mobile robot testbed called Flakey. Our experience indicates the practicality of the complex control methodology, and the ease of incorporating different types of deliberation processes. Examples using the testbed appear throughout the paper (see [Congdon *et al.*, 1993; Saffiotti *et al.*, 1993a; Saffiotti *et al.*, 1993c] for more on our work on Flakey). For reference, Figure 1 gives the main components of Flakey's architecture.

1.5 Plan of the Paper

The next section defines control schemas using techniques from multivalued logic. Their basic properties, including composition and contextual activation, are presented. Section 3 discusses issues of perceptual embedding of the controller, and introduces the concept of *control structure* as a control schema linked with a representation of the environment. Section 4 defines the concept of the goal of a control structure, and presents the main technical result of the paper, the composition theorem. In Section 5 we show how deliberative processes such as planning or goal reduction can be used to generate complex controllers that satisfy high-level goals. Finally, we compare our methods to other approaches in Section 6.

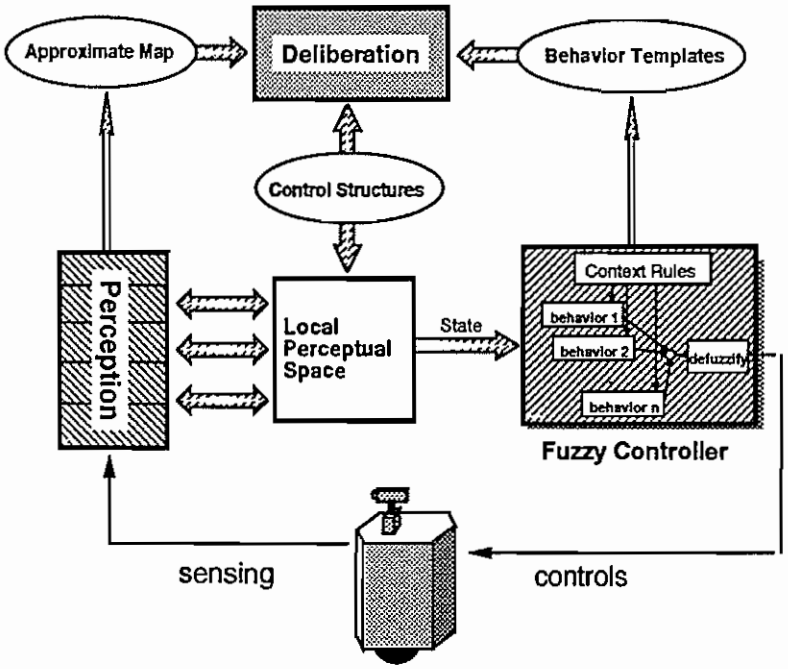


Figure 1: Architecture of Flakey. The local perceptual space is a blackboard-like mechanism for coordinating perceptual routines. Robot motion is produced by a fuzzy controller, whose behaviors are generated by various deliberation processes. Control structures, behavior templates, and the fuzzy controller are all parts of the complex control system.

2 Control Schemas

Control schemas are the basis for physical movements of the agent. Basic control schemas implement basic motor commands; more complex movements are implemented by composing the basic schemas. We describe control schemas by means of *desirability functions*: functions that map internal states of the agent to sets of effector commands, or *controls*, that are most desirable from the point of view of that schema. A control schema implicitly defines a set of possible overall movements, or *trajectories*: all those produced by applying, at every state, a most desired effector command. Trajectories fully characterize the type of movements generated by a control schema. In this section we concentrate on the control of movements *per se*; subsequent sections relate control schemas to action in the world and the goals of the agent, using an analysis of trajectories.

Throughout this paper, we use the framework of many-valued logics to formalize our notions of controls, goals, and behaviors [Rescher, 1967; Rescher, 1969]. As originally suggested by Ruspini [Ruspini, 1991a; Ruspini, 1991b], the ability of many-valued logics to express partial degrees of truth can be effectively used to model the notions of partial satisfaction of goals and utility of control actions, and to formalize the intuition of "good" trade-offs between competing controls and concurrent goals. Before treating control schemas, we need to give some background on our use of many-valued logics.

We take the truth value of a proposition P in a state s, denoted by $\mathcal{T}(P,s)$, to be a real number in the interval $[0,1]$, with 0 standing for complete falsity, and 1 for complete truth. The truth value of a composite formula is given by the following:

$$\begin{aligned} \mathcal{T}(\neg P,s) &= 0 \oslash \mathcal{T}(P,s) \\ \mathcal{T}(P \wedge Q,s) &= \mathcal{T}(P,s) \circledast \mathcal{T}(Q,s) \\ \mathcal{T}(P \vee Q,s) &= \mathcal{T}(P,s) \oplus \mathcal{T}(Q,s) \end{aligned} \tag{1}$$

where $\circledast$ stands for a triangular norm, a type of binary operation widely used in many-valued and fuzzy logics; and $\oplus$ and $\oslash$ are its dual operation and its (pseudo)inverse, respectively [Schweizer and Sklar, 1983]. While relevant to the formalization, the mathematical details of these operations are not essential to the main line of argument of this paper, and we will make things a little simpler by using one specific triangular norm throughout: that is, we let

$$\begin{aligned} x \circledast y &= \min(x,y) \\ x \oplus y &= \max(x,y) \\ x \oslash y &= \min(1, 1+x-y)\,. \end{aligned} \tag{2}$$

These operations define the Lukasiewicz logic $\mathrm{L}_{\aleph_1}$ [Lukasiewicz and Tarski, 1983; Rescher,

1969], and are a common choice in the literature of many-valued logic. These are also the operations we use in our implementation on Flakey.

We will often be interested in the set $[\![P]\!]$ of states where a certain proposition P holds.[2] As in any state s the truth value of P is a number in $[0,1]$, so is the membership of s to $[\![P]\!]$:

$$[\![P]\!](s) = \mathcal{T}(P,s)\,.$$

These generalized sets, whose membership function ranges over the $[0,1]$ interval, are commonly known as *fuzzy sets* [Zadeh, 1965; Klir and Folger, 1988]. Given two fuzzy sets C_1 and C_2, we can use (1) to define the intersection, subtraction, and union of C_1 and C_2:

$$\begin{array}{rcl} (C_1 \cap C_2)(s) & = & C_1(s) \circledast C_2(s) \\ (C_1 \setminus C_2)(s) & = & C_1(s) \circledast (0 \oslash C_2(s)) \\ (C_1 \cup C_2)(s) & = & C_1(s) \oplus C_2(s) \end{array} \tag{3}$$

When we use the norms (2), these operations correspond to the ones originally defined by Zadeh for his fuzzy sets.

2.1 Desirability Functions

A control is generally defined by a function that produces, in any state, a control action to the effectors. When we consider execution in the real world, we need to be less rigid in our definition of movements. 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 control function to be able to generate it. We extend our notion of a control function to produce, in any given situation, a value of *desirability* of each possible control. Intuitively, this value measures how much applying that control is compatible with the performance of that type of movement [Ruspini, 1991a; Ruspini, 1991b].

More precisely, if we take S to denote the set of *internal states* of the agent, and A to denote the set of possible *control actions* available to the agent, we define a *desirability function* from states to actions.

DEFINITION 2.1 *A desirability function is any function*

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

[2] For the sake of simplicity, and where there is no ambiguity, we will sometimes use P to mean both a multivalued formula and the fuzzy set of states defined by it.

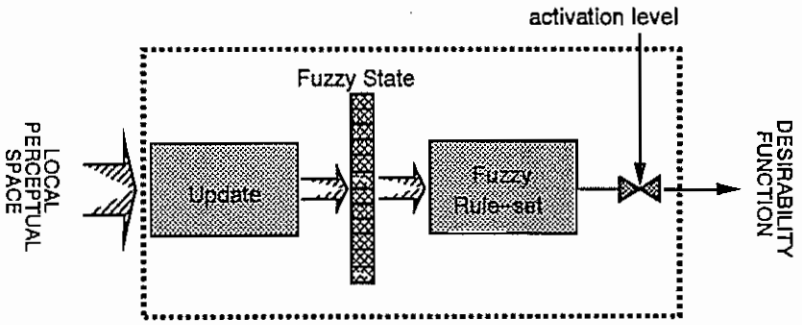


Figure 2: A control schema can be implemented as a fuzzy controller.

from internal states to control actions.

Intuitively, $D(s,a)$ measures the desirability, from the point of view of control schema D, of executing control action a when in state s. In a given state s, D induces a *preference* over possible control actions in the obvious way.

Desirability functions can be encoded in many ways in an artificial agent. One possibility, that we have used in our mobile robot, Flakey, is to use techniques based on fuzzy control [Lee, 1990; Ruspini, 1990; Saffiotti *et al.*, 1993a; Saffiotti *et al.*, 1993c]. Figure 2 shows the structure of a control schema as implemented in Flakey (the "activation level" will be discussed later in this section). Control actions are pairs (α, θ), where α is a linear acceleration and θ is a turning angle. The *fuzzy state* is a vector of fuzzy variables (each having a value in $[0,1]$) representing the truth values of a set of fuzzy propositions of interest, like "there is an obstacle close on the left." At every cycle, the **Update** module looks at the input variables, and produces a new fuzzy state; notice that the fuzzy state corresponds to a fuzzy set of the S state used above.

The **Fuzzy Rule-Set** module contains a set of fuzzy rules of the form

(4) IF P THEN A,

where P is a proposition in our many-valued logic, and C is a fuzzy predicate over control actions: for any possible control action a, $C(a)$ tells how much a is a good instance of C. For instance, the following rule is part of the KEEP-OFF control schema of Flakey:

```
IF obstacle-close-in-front
AND NOT obstacle-close-on-left
THEN turn sharp-left
```

Figure 3 shows the truth value of the "obstacle-close" predicates as a function of the distance d of the closest object detected by Flakey's sensors, and the "sharp-left" predicate as a function of the turning angle.

Rule (4) above encodes a desirability function given, for any state s and control action a, by:

(5) $$D(s,a) = \mathcal{T}(P,s) \circledast A(a)\,,$$

where $\mathcal{T}(P,s)$ is the truth value of P is state s, computed according to (1) above. Consider the obstacle avoidance rule above, and suppose a state s where the sensors detect some obstacle in front at 0.7 meter. Then, we have, for instance:

$$\begin{aligned} D(s,5) &= \min(0,0.8) &&= 0 \\ D(s,7.5) &= \min(0.5,0.8) &&= 0.5 \\ D(s,10) &= \min(1,0.8) &&= 0.8\,. \end{aligned}$$

A control schema normally consists of a set of several rules like (4), each expressing a set of desirable controls to use when the state s satisfies the antecedent. Each rule R_i in the set computes a corresponding desirability function D_i by (5): these are then unioned using the $\oplus$ (max) operator to obtain an overall desirability function $D(s,a) = D_1(s,a) \oplus \ldots \oplus D_n(s,a)$ An interesting problem is what guidance, if any, can the theory give us on how to generate sets of rules that produce a given control. This problem is a subject of our current study [Ruspini and Saffiotti, 1993].

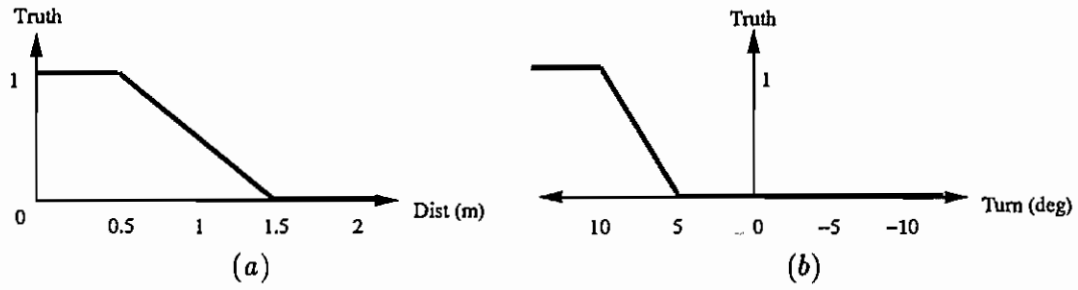


Figure 3: The fuzzy predicates "obstacle-close" (a), and "sharp-left" (b).

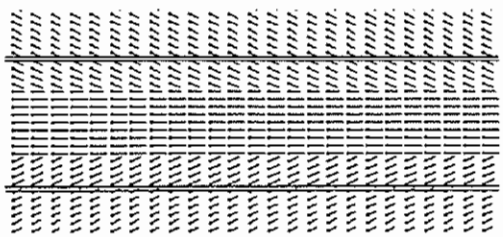


Figure 4: A vector field showing the preferred control for the FOLLOW schema. Direction is left to right. A small dot indicates the origin of each vector.

However it is produced and implemented, a desirability function specifies, for each state, a ranking over possible controls — it does *not* specify a unique control to apply in that situation. In order to *act*, the agent needs to use this ranking to *choose* one specific control action $\hat{a}$ for actual execution. Said differently, the (fuzzy) class of movements described by the desirability function D needs to be instantiated into one specific element. In the case of Flakey, where continuous control values have to be traded off, we use the following formula:

$$\hat{a} = \frac{\int a \cdot D(s,a)\, da}{\int D(s,a)\, da} \tag{6}$$

that computes the mean of the possible control actions weighted by their degree of desirability. Equation (6) is known in the fuzzy control literature as "centroid defuzzification" [Lee, 1990]. The use of (6) has been found satisfactory in our experiments, provided that the rules in a control schema do not suggest dramatically opposite actions in the same situation: when this happens, averaging using (6) simply does not make sense. (The best tradeoff between avoiding an incoming train from the left or from the right is seldom to stay on the rails!) Our empirical strategy has been to make sure that the premises of conflicting control rules are not applicable to the same situation; other authors have used alternative choice rules (e.g., [Yen and Pfluger, 1992]).

Figure 4 shows the preferred control actions for Flakey's FOLLOW control schema: for each point of the grid, the vector shows the chosen direction and speed obtained by applying the rules of FOLLOW and the (6) criterium. The intended effect of this control schema is to move inside a given "lane" (marked by the double lines) at a fixed speed. Flakey has a dozen other control schemas, including schemas for crossing a door, for reaching a near location, and so on; each schema typically consists of four to eight rules.

2.2 Trajectories

To get a more global view of what movements a control schema can produce, we introduce the notion of a *trajectory*. A trajectory is a specific sequence of states in which the agent can find itself as a result of repeatedly applying a control scheme. In general, a control schema can generate a large number of trajectories: for example, my hand can follow an infinite number of paths when I stretch my arm. All these trajectories, however, will have something in common — that *quid* that identifies the particular type of movement generated by that control schema. Studying this *quid* means finding the properties of a control schema, and is a necessary step to linking the execution of control schemas in an environment to their effects.

We need one more ingredient: a description of the effects of the actions available to the agent. We assume that we are given a function

$$M : S \times A \times S \rightarrow [0, 1] \tag{7}$$

such that $M(s, a, s')$ measures how much s' can be the state resulting from an agent's executing action a in state s. Notice that M may represent weak information: given a state and an action, M provides us only with an elastic constraint as to what the next state might possibly be. In particular, we are not assuming to have any probability distribution available regarding the effects of actions. As we are interested in agents operating in real environments under conditions of uncertainty and incompleteness, this freedom seems to be a necessary element. More constrained cases can be captured by imposing specific conditions over M — for example, the case where there is no uncertainty about the effects of actions is captured by letting, for any s and a, $M(s, a, s') = 1$ for exactly one s'.

Another important point to notice is that M is expressed in terms of an agent's internal states, not in terms of the actual effects on the real world. In a sense, we are considering here the effects of actions only as perceived by the agent. This agent-centered perspective is ubiquitous in our approach, and we will say more about it in Section 3.[3]

Knowing how the agent moves from state to state in response to its actions, we now measure the possibility that a certain control schema D will produce a transition from a

[3]Although our framework can be translated into the concept and notations more conventionally used in control theory (e.g., [Bellman, 1961]; see also [Dean and Wellman, 1991] for a more AI view), the perspective we adopt is different. While the latter focuses on an "external" view whose main objects are the states of the overall *agent + environment* system, linked to the internal state of the agent through the agent's perception, we focus first on the internal state of the agent, and later analyze the link between this state and the external world. In addition, we make no explicit reference to time: time is only implicitly encoded into the states in S.

state s to a state s'. To do this, we consider the actions that can produce this transition according to M, and look at how desirable these actions are according to D. Formally, we define a function $Next_D : S \times S \to [0,1]$ as follows:

$$Next_D(s,s') = \sup_{a \in A} \left(D(s,a) \circledast M(s,a,s') \right) \tag{8}$$

We are now ready to use trajectories to characterize control schemas. We take a trajectory to be any element of S^ω, that is, any finite sequence

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

of states in S,[4] and ask ourselves the question: "what are the trajectories t that can possibly be generated as a result of applying D?" We call these trajectories *desirable* for D. The degree to which each trajectory is desirable for D is measured by the following function:

DEFINITION 2.2 (DESIRABLE TRAJECTORIES)

Let D be any desirability function. Then $Traj_D$ is the function $Traj_D : S^\omega \to [0,1]$ defined, for any trajectory $t \in S^\omega$, $t = (s_0, s_1, \ldots, s_k)$, $k > 0$, by:

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

It is important to have a clear idea of what it means for a trajectory t to be desirable for D. From the definition, the level of desirability of t depends only on the value of the least desirable step in the entire trajectory. This is essentially a local measure: one trajectory whose worst move has desirability 0.4 for D, while all the other moves have desirability 1, is exactly as desirable as one trajectory whose all moves have desirability 0.4. Thus, desirability should not be confused with the "quality" of a trajectory in any sense: we will be concerned with the quality of a control schema only when we deal with the link between controls and goals in the next section. As goals are global objects, this link is the key to ensuring that the choices made locally by a control schema are compatible with the global objective expressed by the goal.

A second point to keep in mind is that desirability of trajectories refers only to the agent's movement capabilities, abstracted from any environment. In particular, it is not guaranteed that the trajectories desirable for a control schema will be *physically admissible*

[4] More generally, trajectories are elements of $S^\omega \times A^\omega$, for what action has been performed in the transition from s to s' may be relevant. For ease of notation, we consider only trajectories in S^ω here: our treatment immediately extends to the general case.

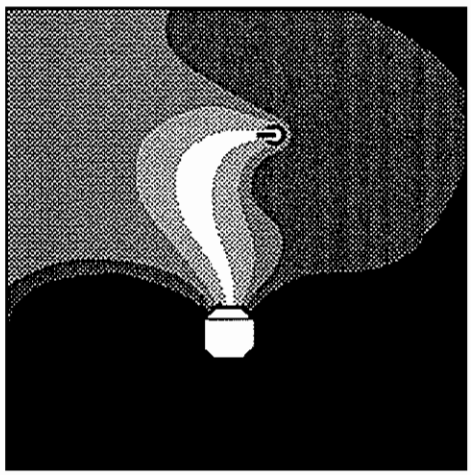


Figure 5: Some families of trajectories generated by GO-TO-CONTROL-POINT. Fading from black to white indicates increasing level of desirability.

in the particular environment where the agent is embedded: for instance, a river may block most of the desired trajectories. To account for physical admissibility, we suppose that we are given an *admissible state function*

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

such that $E(s)$ tells, for any state s, how much being in s is admissible, and we give the following definition for admissible trajectories.

DEFINITION 2.3 (ADMISSIBLE TRAJECTORIES)

Let D be any desirability function, and E an admissible state function. Then Adm_D is the function $Adm_D : S^\omega \to [0, 1]$ defined, for any trajectory $t \in S^\omega$, by:

$$Adm_D(t) = Traj_D(t) \circledast \inf_{s \in t} E(s)$$

For instance, a robot may consider a state where it is partially inside a wall as not admissible; hence, any trajectory that, at some point, includes a passage through a wall will not be admissible. We will come back to the issue of admissibility when considering the embedding problem in the next section.

Figure 5 gives a pictorial representation of some trajectories for our robot Flakey. Flakey has a control scheme called GO-TO-CONTROL-POINT aimed at reaching a given location

in Flakey's workspace from a given angle. In the pictures, Flakey is in the lower half of the space, in top-view; the target location is the semicircle in the upper half, with a tail indicating the entry direction. The picture shows some of the desired trajectories: for each family of trajectory, the level of gray indicates the level of desirability, with white standing for 1 and black for 0.

2.3 Complex Controls

An agent is often engaged in activities requiring the simultaneous activation of several basic control schemas, and/or in multiple activities. For instance, a robot may be going down a hallway while avoiding the obstacles on its way and moving its camera to keep track of some landmark. In any state s, each one of these concurrent control schemas provides a measure $D_i(s,a)$ of the desirability or each action a from its own point of view. We are interested in finding a desirability function D that models the combination, or *blending,* of these control schemas. In what follows, we look at various ways in which control schemas can be combined, and characterize the resulting complex controls in terms of sets of admissible trajectories.

A first type of combination is one where the desirability functions are considered conjunctively: at every state, the combined desirability of a control is given by the minimum (or, in general, $\circledast$) desirability of that control from the point of view of each schema:

$$(D_1 \otimes D_2)(s,a) = D_1(s,a) \circledast D_2(s,a)\,. \tag{9}$$

We call $(D_1 \otimes D_2)$ the *conjunctive combination* of D_1 and D_2. A disjunctive combination can be defined in a similar way using $\oplus$.

Conjunctive combination works well for combining control schemas that are not conflicting; that is, it is not the case that, in some state, all the desired controls for one schema are undesired for the other, and *vice-versa.* When this happens, the combined desirability function would be meaningless: in the conflicting states, it would identically assign zero desirability to any possible control. The key observation here is that each control schema has in general its own *context* of applicability. Correspondingly, we would like each component desirability function to be considered only when appropriate, by weighting the impact of the control actions it suggests according to how much its context is the case in the present state. For example, we may want to combine a control that guides a robot along a hallway when no obstacle is detected, and a control that keeps it away from obstacles when there is a risk of collision, to obtain a control to safely traverse a corridor cluttered with obstacles. In the state where the robot is facing an obstacle, the first schema would prefer controls

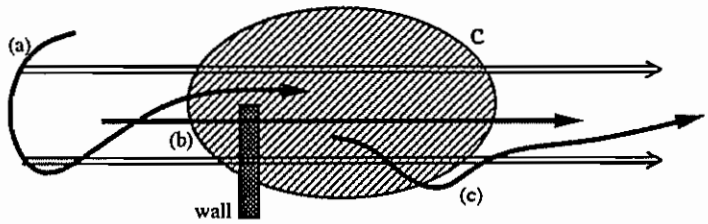


Figure 6: The restriction of the FOLLOW control schema to the context C. (a) and (b) are both desirable, but only (a) is admissible; (c) is not desirable.

that keep a cruise speed, while the second one would favor controls that slow down the robot. By considering the context, we recognize that the first schema is outside its area of competence, and discard its desires.

We represent a context by a fuzzy set C of states of S.

DEFINITION 2.4 *A context C is any function*

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

from internal states to truth values.

For any context C, we can define a *restriction* operator $^{\downarrow C}$ that maps desirability functions into desirability functions in the following way:

$$D^{\downarrow C}(s, a) = D(s, a) \oslash C(s) \,. \tag{10}$$

We call $D^{\downarrow C}$ the restriction of D to C. Recalling the definition of the particular $\oslash$ we are using, we can read (10) as follows. If a state s is completely out of context, then any control can be used indifferently: that is, if $C(s) = 0$, then $D^{\downarrow C}(s, a) = 1$ for any a. Inside C, the desirability of controls is given by D: that is, if $C(s) = 1$, then $D^{\downarrow C}(s, a) = D(s, a)$. And in intermediate situations, that is, when $0 < C(s) < 1$, D is considered, but other controls are also partially admitted. In other words, restriction leads to a less committal control: one that expresses a preference only when the agent is in one of the context states, but allows anything outside these states. In particular, this implies that the desired and admissible trajectories for $D^{\downarrow C}$ are trajectories that satisfy D inside C, and are totally free outside C. Figure 6 shows three trajectories in the context of the control schema

$$\text{FOLLOW}^{\downarrow C}$$

executed in an environment with an inconvenient wall. Trajectories (a) and (b) are fully *desired*, as they both run in the middle of the target lane and parallel to it whenever in C; however, as it crosses the wall, (b) is not *admissible* in this environment. Trajectory (c) is not desirable, as it fails to follow the lane while in C.

We can extend the notion of *being in a context* from states to trajectories by letting, for any context C and trajectory t:

$$C(t) = \inf_{s \in t} C(s) . \tag{11}$$

The notion of context will be pivotal to the development of our theory of behaviors in the rest of this paper: the basic intuition is to restrict the zone of influence of a particular control schema to those situations where it is competent, and give way to other schemas in other situations. This general pattern of combination of control schema, combining context restriction and conjunctive combination, is summarized in the following definition.

DEFINITION 2.5 *The* context-dependent blending *of D_1 and D_2 under C_1 and C_2 is given by:*

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

The result of Definition 2.5 is a desirability function that considers, at any state s, the desirability expressed by D_1 weighted by how much s satisfies C_1, and the desirabilities expressed by D_2 weighted by C_2.

Context-dependent blending can be implemented in the fuzzy controller by introducing meta-level rules of the type

IF C THEN *activate* D (12)

where D is a control schema and C a context; this is the solution we have adopted in Flakey. More precisely, the truth value of C in the current state is used to discount the output of the fuzzy controller for the schema D — this is the use of the "activation level" input in Figure 2 above. Many such fuzzy controllers can simultaneously run in Flakey, each one implementing one schema's desirability function. All these desirability functions are discounted by the corresponding context, and then merged into a composite one by Definition 2.5, that is, by computing the pointwise maximum. Finally, the defuzzification formula (6) is used to convert the resulting tradeoff desirability into one crisp control action.

Figure 7 shows a simple example of context-depending blending where a FOLLOW control schema is used to follow a corridor, and a KEEP-OFF schema is used to stay away from obstacles. The meta-rules used to encode the contexts in this example are:

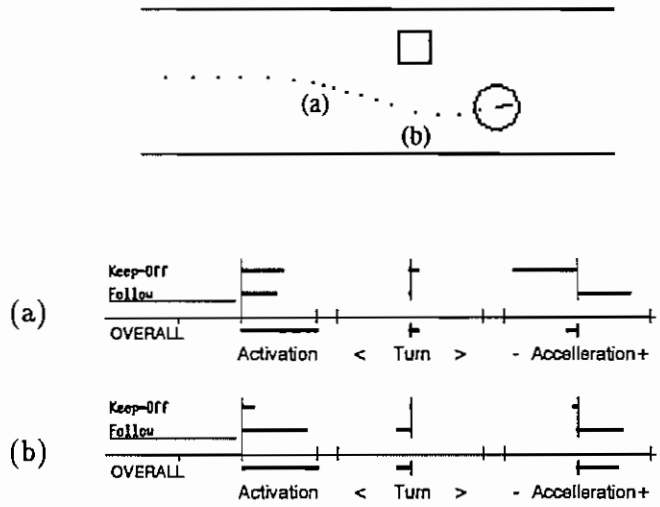


Figure 7: Context-depending blending of corridor-following and obstacle-avoidance schemas.

```
IF approaching_obstacle THEN activate Keep-Off
IF NOT approaching_obstacle THEN activate Follow
```

The bars in the picture show the level of activation and the preferred controls (turn and acceleration) for each schema, and the result of the blending. In (a), an obstacle has been detected by Flakey's sensors, and the preferences of KEEP-OFF are dominating; later, when the path is clear, the preferences expressed by FOLLOW regain importance (b), and Flakey returns to the middle of the hallway.

We end this section by giving a couple of theorems that characterize the control schemas obtained by conjunctive combination and by restriction in terms of the trajectories they generate. These results show that our combination operators "behave well" with respect to the admissible trajectories of the component control schemas. The proofs are given in the Appendix.

THEOREM 2.1 *Let D_1 and D_2 be two desirability functions, and t any trajectory in S^ω. Then*

$$Traj_{(D_1 \otimes D_2)}(t) \leq Traj_{D_1}(t) \circledast Traj_{D_2}(t)$$

THEOREM 2.2 *Let D be a desirability function, and C a context on S. Then*

$$Traj_{D|C}(t) \leq \sup_{t' \subseteq t} [\, Traj_D(t') \oslash C(t') \,]$$

Intuitively, Theorem 2.1 says that the trajectories that can be produced by a conjunctive combination of behaviors are bounded by their admissibility from the point of view of each of the conjuncts; and Theorem 2.2 says that the trajectories generated by a restricted behavior $D^{\downarrow C}$ are at most as admissible as than those whose intersection with C (i.e., the subtrajectories t' of t formed only of states that are in C) is admissible for D (and are uncommitted elsewhere).

The reader can use these results to verify that the admissible trajectories of the context-dependent blending (2.5) are exactly what is intuitively expected — they "follow" D_1 when they are inside the context C_1, follow D_2 inside C_2, and constitute a tradeoff between D_1 and D_2 in the common context $C_1 \cap C_2$. In more abstract terms, context-dependent blending provides a handle to fuse control schemas in a state space subdivided into (possibly overlapping) "areas of competence".

3 Embedding and Behavior

The purpose of the controller is to produce action in the world that satisfies the goals of the agent; unless the controller accomplishes this, it is not successful. But the controller itself only references the internal state S; nothing is specified about the relation of the internal state to the world, or to the behavior of the agent. For example, the controller may issue signals for the agent to move forward; but it is only in the context of facing an open doorway to a room that the agent will accomplish the action "move into the room." Thus, it is only when the controller is embedded (or, to use a trendier term, *situated*) in an environment that the controller acts to produce behavior (see Figure 8). For the designer of a system, it is this behavior that is important; the designer wants a system that will act to satisfy certain goals, which are naturally expressed as states or histories of the environment. The key is to create a controller that, given an environment, will result in the intended behavior. In this section we examine issues of embedding the formal controller into the real world as a prelude to analyzing the behavior produced by the controller in terms of more abstract goals.

Our approach to embedding is model-based: we use a perceptual subsystem to tie representations of objects in the internal state to their counterparts in the real world. These representations are called *artifacts*. The perceptual subsystem methodology is typical of AI planning systems that talk in terms of actions, for example, the operator schema *pickup*($?A$). Such systems avoid the problem of perceptual anchoring by specifying operators in terms of parameters that refer to real-world objects, and assuming that there is a perceptual subsystem capable of picking out the correct block. They also avoid the problem of instantiating actions as a sequence of physical movements, assuming that there is a controller capable of this task.

What is unusual here is that we are employing the model-based perceptual technology in designing a control system. Typically, embedded controllers use raw sensor readings as direct input to a closed-loop feedback system. A model of sensor response and the system dynamics may be part of the controller, but there is no attempt to abstract to the level of actions and objects typically used by planning systems. We have found that, through the use of model-based perception, we can decouple the action of the controller from the problems of interpreting noisy sensor data. The controller has a closed-loop response to the environment, but it is mediated by the perceptual system. Hence, the controller acts with respect to artifacts, and its actions can be described in the abstract terms necessary to link the controller to planning and other decision-making systems.

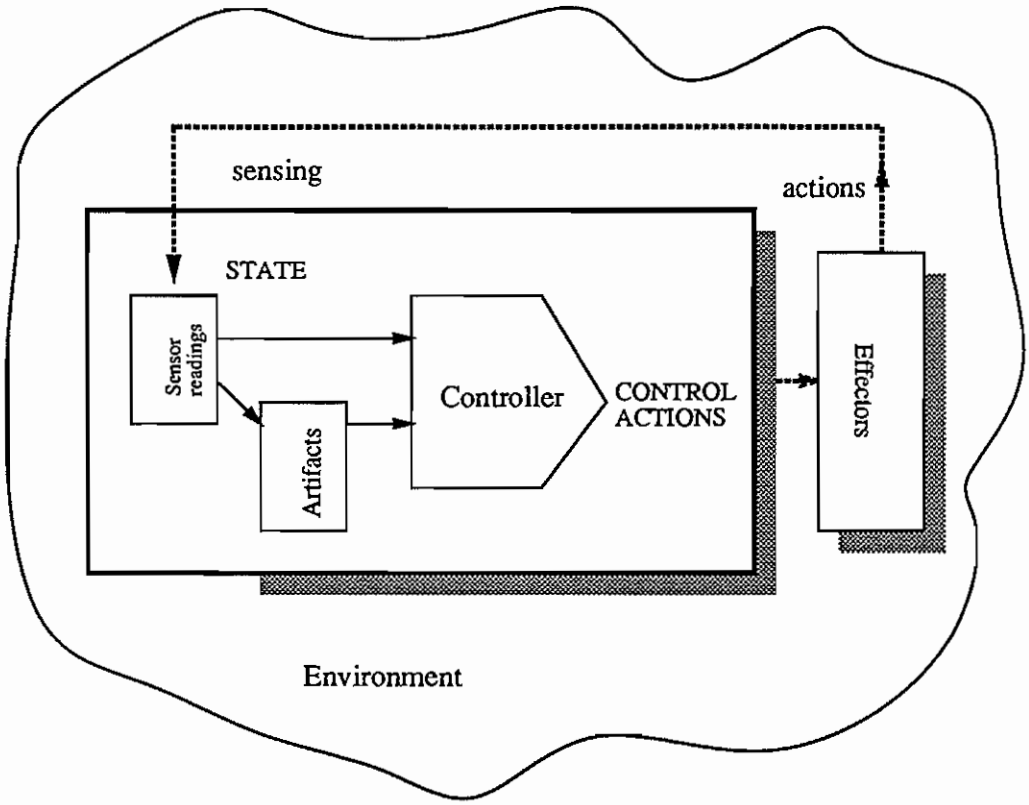


Figure 8: An embedded controller. The control rules operate on formal input and output variables. The input variables are grounded or anchored to the environment through the sensors and interpretation by perceptual routines. Control output variables produce actions whose effects can be measured by perception.

In this section, we define *control structures* as a means of embedding control schemas into an environment. We then discuss properties of embedding and the perceptual subsystem, especially the problem of keeping the controller reactive to new or unforeseen information. Finally, we describe several methods by which control structures can help with the problem of keeping the internal state anchored to the environment.

3.1 Control Structures

We define control structures (CSs) as a means of linking control schemas to behavior in the environment. Control structures incorporate a desirability function representing the formal controller, an *artifact* that is the internal representation of some object in the environment, and a context for execution of the CS. It is the task of the perceptual subsystem to keep the artifact properly registered, or *anchored*, with its corresponding object. Assuming that this is the case, the outputs of the controller can be "lifted" from the formal domain to trajectories in the environment, and we can describe the embedded controller in terms of its external behavior.

Control structures bridge the gap between abstract action descriptions and physical control of movement [Firby, 1989; McDermott, 1990b]. They are inherently movement-oriented, since the desirability function induces a preference on control actions of the agent. But they also incorporate elements of abstract action: objects that the action operates on (the CS artifact), and preconditions for the success of the action (the CS context). The third component, the postcondition or goal of the action, can be specified as a predicate on trajectories produced by the CS, and is considered in Section 4.

Formally, we define control structures (CS) as follows:

DEFINITION 3.1 (CONTROL STRUCTURES) *A* control structure *is a triple*

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

where $C : S \rightarrow [0,1]$ *is a fuzzy set of states,* $D : S \times A \rightarrow [0,1]$ *is a desirability function, and* O *is a set of individual constants.*

Intuitively, O is the set of artifacts relevant to the control schema D, and C is a context for the control. D is a control schema taking the artifacts as input and producing control actions (Definition 2.1). C is a context for D (Definition 2.4); it encodes the situations where the control structure is relevant. Hence, a control structure can be thought of as a specification of *what control schema* should be used with respect to *which object* and under

which conditions. Executing this control structure is our basic type of action: we call this action the *behavior* induced by the control structure.

The artifacts are identifiers of internal variables representing the aspects of the state that are relevant to a particular CS. Artifacts do not need to correspond to physical objects in the environment: an imaginary spot to reach can be an artifact. However, though imaginary, the artifact must be situated in the world, if it is to be used as an object of action. Some artifacts can be as abstract as "the configuration of occupied space around me" — for example, the obstacle grid OG artifact used below. We discuss artifacts and perception in more detail in the next subsection.

The context has many different uses. It can specify state conditions that must be satisfied for the CS to achieve a particular goal, like the preconditions of action schemas in traditional planners. Typically, it will give conditions for the anchoring of the CS's artifact; some examples of this are in the next subsection. It can also be used to coordinate the action of concurrent CSs: Section 4 discusses how contexts are used in composing CSs.

To make all this more concrete, consider a CS aimed at having a robot move down a given corridor:

(13) $$\langle at(\text{Corr-A}), \text{Follow}(\text{Corr-A}), \text{Corr-A} \rangle .$$

The control relation Follow consists moving at a certain speed along the centerline of the corridor; Figure 4 shows a vector field corresponding to this relation: for each point of the grid, the vector shows the most desirable orientation and speed of the robot given by Follow. The artifact Corr-A consists of the two sets of double lines. It is a representation of an abstract corridor, whose initial position is based on prior knowledge. As the robot proceeds, perceptual routines are used to sense the real walls, and to continually update Corr-A's position accordingly. Figure 9 is an example of this process. The double lines are the corridor artifact; Follow keeps the robot traveling in the approximate direction of the corridor (a). When enough sensor readings have been gathered to identify the exact position of the corridor, the artifact is updated (b). Hence, Corr-A is kept *anchored* to the real corridor whenever the perceptual data are available, while acting as an assumption otherwise (e.g., if the wall is momentarily occluded by an obstacle). Notice that, through anchoring, an abstract type of movement, "proceed between two internally represented rails", has been raised to the level of a *behavior* in an environment: "follow a given corridor". Executing the same movement with a different anchoring in a different environment could result in a different behavior, for example, "cross a ballroom."

This example illustrates two points of special interest. First, the use of the corridor wall artifact enables the controller to produce useful movement even when the sensor readings

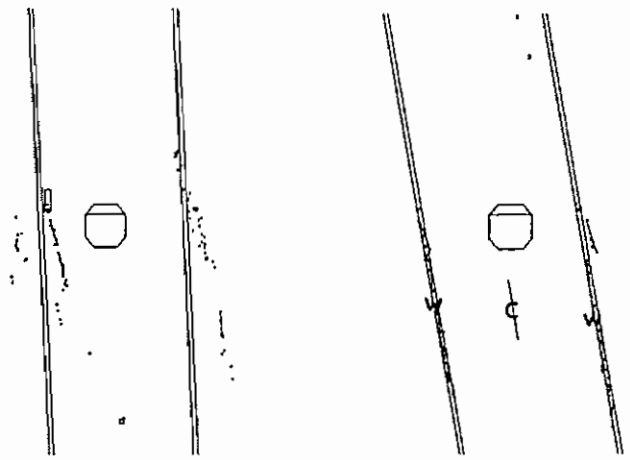


Figure 9: Before anchoring (a), the corridor artifact approximates the position of the corridor. After enough sensor readings are available to make a perceptual identification, the position of the artifact is updated (b), with the walls and center of the corridor identified.

are temporarily obscured or unreliable. In control theory terms, the CS acts like an open-loop controller in computing control actions with respect to the artifact; the perceptual process closes the loop from time to time by updating the position of the artifact relative based on perception.

Second, depending on the exact shape of the corridor and the initial position of the agent, the Follow CS produces an infinite variety of physical movements, all of which satisfy the goal of moving down the corridor [Israel *et al.*, 1991]. That is, the CS is a controller of physical movement that can be interpreted as a behavior at a higher level of abstraction, suitable for use in planning or other deliberative processes.

Of course, the use of artifacts and control structures does not obviate the problems of "closing the loop" between sensors and artifacts, that is, the problems inherent in model-based perception. Still, it gives a framework for addressing fundamental problems of perception and action. While we cannot hope to devise a complete solution here, we have used the strengths of the control structure methodology to give partial answers to several difficult issues.

Reactivity A control structure should use just the information it needs to perform its task, without placing an undue computational burden on the perceptual system. This is accomplished by means of a Local Perceptual Space (LPS), a representation of space

in which perceptual information at different levels of abstraction can be registered.

Abstraction A control structure should identify relevant real-world objects in a manner that is suitable for connecting to a more abstract planning level. Our idea here is to use a description as part of the identification of an artifact. The description contains information that can be used to match the artifact to a real-world object in a manner that is appropriate for the intended behavior of the control structure.

Sensitivity Finally, a behavior must be sensitive to the anchoring of its artifact. The more the artifact is out of alignment with its real-world counterpart, the less the CS will implement the desired behavior. We introduce CSs whose primary task is to keep the agent situated in a way that allows the perceptual system to gather relevant information efficiently. These CSs contribute to *active perception* on the part of the agent [Bajcsy, 1988].

3.2 Anchoring with the Local Perceptual Space

To anchor state variables to objects in the external world, we assume that a perceptual subsystem reliably coordinates internal variables of the control system (artifacts) with external objects. Then, internal control descriptions can be translated directly into external trajectories (behaviors) and goals of the agent, as in the Follow CS just introduced. The process of keeping internal control variables coordinated with the environment is called *anchoring.*

In practice the idea of embedding the control system by means of a perceptual process must be modified to take into account the problems and restrictions associated with the process. 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. Further, particular behaviors may need only part of the full perceptual information available, and the control system should take advantage of this with a tighter coupling to the perceptual process: the perceptual process should be *relevant* to the desired behavior. These criticisms, among others, were part of the thinking that led to the subsumption architecture of Brooks and his colleagues [Connell, 1990; Brooks, 1982]. We have adopted and modified some of these ideas to achieve an embedded control system that is highly reactive, yet preserves the "lifting" feature of the more traditional perceptual processing approach.

Coordination of artifacts is done in the Local Perceptual Space (LPS), a blackboard-like mechanism for keeping track of sensor information and its interpretation in the immediate vicinity of the agent. Geometrically, the LPS is an egocentric cartesian plane in which

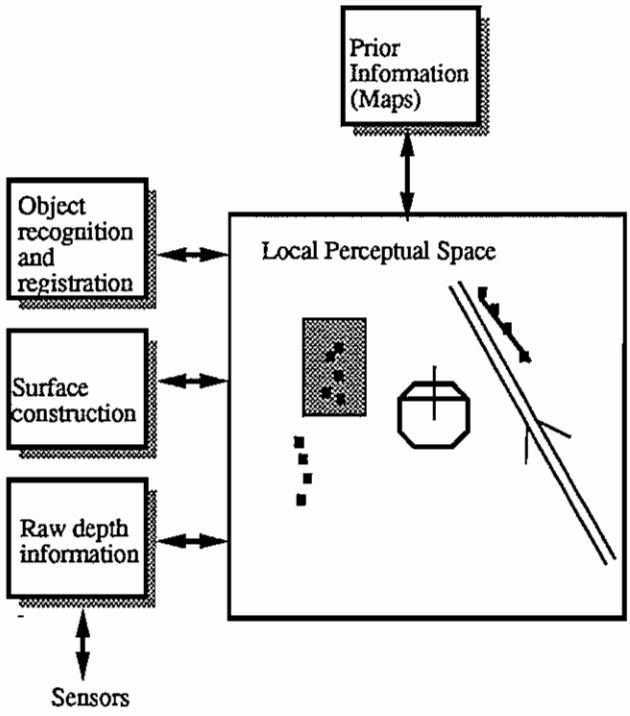


Figure 10: Flakey's system architecture. Perceptual routines are on the left. The vertical dimension gives an indication of the cognitive level of processing, with high-level perceptual routines at the top. Higher-level modules use information generated by lower-level ones. Control structures use information appropriate to their needs.

all sensor information is registered, and various artificial constructs (*artifacts*) are entered. The diagram of Figure 10 shows the major components of the perceptual subsystem. Raw sensor information is received and placed into the LPS with relatively little processing, so it is available quickly. In this case, the small dots are unfiltered sensor readings that indicate the possible presence of objects. In this respect the LPS acts like an occupancy grid, and its information can be used directly by some control structures. For example, the input variables to an obstacle-avoidance CS are various areas of sensitivity that it checks for the presence of readings, as in the rectangular patch in the upper left portion of Figure 10. Using such variables, a CS can be made sensitive to rapidly changing conditions with relatively little perceptual processing. However, because the interpretation process is only cursory, the anchoring to the environment is suspect. The behavior of an obstacle-avoidance CS relying on raw sensor readings will not be reliable, because the readings do not reliably

indicate the presence of obstacles in the environment.[5]

Further processing filters and groups these readings into surface information, as in the linear structure in the upper right of Figure 10. Surface representations contain more useful information than uninterpreted sensor readings: for example, a CS could follow a course parallel to a given surface. Finally, surfaces are grouped into recognized objects, such as the walls of a corridor, represented by the double-shafted arrow on the right of Figure 10. Most purposeful behaviors, such as Follow, are implemented by CSs that use identified objects as their artifacts. The fact that these CSs perform their intended behavior rests on the identification of an internal variable in the LPS (an artifact) with a real-world object.

Although typically artifacts will be generated in a bottom-up fashion from sensor data, they are a general, flexible means of providing input data for control schemas. For example, suppose we want to implement a "foray" behavior that takes the robot out perpendicular from a wall for 5 meters, and then back again. We could write a control structure that uses the wall as its artifact, and keeps track of the geometrical relation between the robot and the wall in the appropriate way. But it is simpler to create a new artifact that is a virtual "wall" perpendicular to the original wall, and then use the previously defined wall-following control schema with the new artifact. By choosing artifacts and control schemas carefully, it is possible to program a large variety of behaviors with a few simple building blocks. We used this capability extensively in programming forays into the center of the arena at the 1992 NCAI robotics competition [Congdon *et al.*, 1993].

A second use for artifacts is anticipation based on prior knowledge. Consider the problem of turning from one corridor into another. As the robot approaches the end of the corridor, before it can see the other corridor, it should start to turn and perhaps slow down. The method of control schema blending can accomplish the transition from one corridor-following CS to the next, but it relies on the artifact, the second corridor wall, being available before the sensor actually sees it. Since the CS relies on a corridor-wall artifact, this artifact can be placed in the LPS at approximately the correct place, based on prior knowledge. The second corridor-following CS acts according to the anticipated but unseen corridor. When sensor information is finally available, the position of the corridor artifact is updated, and the CS adjusts accordingly.

[5]It should be possible to treat the issue of reliability of CS behaviors by examining the performance of the perceptual routines. Such a treatment would be a useful extension of the work in this paper.

3.3 Object Descriptions

So far we have just assumed that the perceptual system knows how to coordinate a given artifact with its real-world counterpart. In this section we develop an abstract implementation of the anchoring process that can be useful in planning. Imagine, by way of example, that a friend asks you to go and retrieve a piece of equipment with which you are not familiar. He would give you a description of the object in terms of properties that (he believes) you can use to identify the object, like its shape, size and color, plus some properties necessary to carry out the action, like the object's approximate location and how to handle it. You will then have to use your perceptual capabilities to: (1) *find* the object, by matching what you are perceiving with the description you have been given; and (2) *acquire* new (or more precise) information about the properties of this object that are relevant for the action, like its exact position and orientation with respect to you.

Formally, this two-step process can be accomplished by specifying *matching* and *action* properties for an object. Consider the action of applying an instance of the control schema A on the n objects denoted by the individual constants $o_1, \ldots, o_n$; we informally denote by $A(o_1, \ldots, o_n)$ this action. We associate to each individual o_i an *object descriptor* $\delta(o_j)$ consisting of a pair of sets of properties:

$$\langle \textit{Match Properties}, \textit{Action Properties} \rangle ,$$

where each property is denoted by a term of the form $\lambda x.\alpha(x)$ with α a first order formula where x occurs free. The match properties are relevant to the identification of the anchoring object, and are meant to guarantee that the object on which the action takes place has the right characteristics for the action: If "pick-up(A)" is meant to pick up a red cube, $\lambda x.\text{color}(x, \text{red})$ will be in the descriptor for A. The action properties are relevant to the actual execution of the action — the (assumed) location of the object may form one such property: $\lambda x.\text{at}(x, \vec{l}_0)$. The controller uses these properties in controlling the physical movements. The match properties and the action properties are normally instantiated by a planning system on the basis of the agent's knowledge about the environment and the agent's goals, respectively.

To make things more concrete, consider the example of anchoring the corridor wall in Figure 9. Let $\hat{x}$ be a variable representing the corridor artifact; it is associated with the

following object descriptor:[6]

(14)

Match Properties	*Action Properties*
corridor($\hat{x}$)	at($\hat{x}$, (+150, −150))
near($\hat{x}$, 150, 20)	oriented($\hat{x}$, 0°)
angled($\hat{x}$, 0°, 20°)	

The action properties give the current estimated position of the walls of the corridor on each side of the robot (+150, -150), and their relative orientation. The match properties are estimates of the uncertainty in the actual corridor, based on knowledge of the robot dynamics and prior knowledge. Each wall should be within 150 ± 20 units of the robot, and should not be angled more than 20° from vertical.

The action properties are used by the controller to generate movement; the matching properties are used by the perceptual subsystem to find and verify surface-feature candidates for the walls of the corridor.

When the perceptual subsystem matches the corridor artifact to sensor input, it generates something like

(15) $$\text{corridor}(t_1) \wedge \text{at}(c_1, (+154, -142)) \wedge \text{oriented}(c_1, -4°)$$

where t_1 is a new Skolem constant used to denote the newly perceived object. The conjuncts in (15) match the properties in (14), and the agent *anchors* $\hat{x}$ to the perceived object t_1, and updates the object descriptor for $\hat{x}$ as follows:

(16)

Match Properties	*Action Properties*
corridor($\hat{x}$)	at($\hat{x}$, (+154, −142))
near($\hat{x}$, 150, 10)	oriented($\hat{x}$, −4°)
angled($\hat{x}$, −4°, 5°)	

The controller's artifact now reflects the perceived physical object with less uncertainty than before.

3.4 Behaviors for Perception

To operate efficiently, perceptual processes must be sensitive to the needs of the controller, and the controller should formulate its activity with the requirements of perception in mind. The feedback between perception and control comes under the general heading of "active

[6]For sake of simplicity, we write $P(o_j, t_1, \ldots, t_k)$ for $\lambda x.P(x, t_1, \ldots, t_k)$ in the object descriptor associated with o_j. Later, we use $P(t, t_1, \ldots, t_k)$ in a similar way as an abbreviation to indicate that a perceptual property $\lambda x.Q(x, t_1, \ldots, t_k)$ holds of an individual t.

perception." Since control structures contain an artifact that must be kept anchored, they provide a means both for directing the action of the perceptual subsystem, and signaling when their behavior is inappropriate because the artifact is unanchored. Consider the control structure for following a corridor (13). The anchoring of the corridor artifact is critical to its behavior; hence, whenever it is instantiated, it signals the perceptual subsystem to institute a process that tries to keep the artifact anchored. The anchoring is an implicit part of the context of the behavior; if it should fail, the behavior is no longer appropriate.

In addition to keeping track of the anchoring of their artifacts, behaviors should actively try to maintain the anchoring by moving in a way helpful to the perceptual process. For example, in the Follow behavior, 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. Under normal circumstances the Follow control schema will move the robot steadily enough to gather the required information; but a corridor crowded with obstacles could interrupt the readings. To remain anchored in these situations, the Follow behavior is paired with a Sense behavior:

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

As its corridor artifact becomes unanchored, the Sense behavior slows down the robot and orients it in the most likely direction of the corridor. Once the corridor is recognized by the perceptual subsystem, the context becomes false, and the Follow behavior can proceed normally. The two behaviors are blended using the mechanisms described in Section 4.

This is a simple strategy, and in situations with complex obstacles may not work. If the corridor artifact stays unanchored, 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. An example of such a decision process is in Section 5. The advantage of using sensing behaviors that act to maintain an anchoring is that they are simple and can be extremely reactive to the current situation.

4 Goals

Control structures, by virtue of the anchoring of their artifacts, produce trajectories that depend on objects in the environment. A control structure is designed with some purpose in mind, that is, it is designed to produce trajectories that satisfy some goal of the agent. In this section we consider how the goals of a CS can be specified formally, as a prelude to linking control structures to more abstract operator formulations of agent action. In addition, we prove the main formal results of the paper, two composition theorems: the *conjunctive composition* of two control structures is a complex control that satisfies the conjunct of their goals; and the *chaining composition* of two control structures is a complex control that satisfies the goal of the second one in a wider context.

The goal of a CS can be any predicate that is true of any trajectory of the CS in an environment that satisfies its context. Typically, a CS will have a single goal, for example, the AVOID CS has the goal of not running into obstacles. But a CS may also have an infinite number of useful goals. For example, a CS using Follow can move the robot to an arbitrary point p along a corridor:

$$\langle at(\mathsf{corr}) \wedge \neg at(p), \mathsf{Follow}, \{\mathsf{corr}\} \rangle \, .$$

We take a *goal* to be any function

$$G : S^{\omega} \rightarrow [0,1] \, . \tag{18}$$

That is, a goal is a function that indicates how much each trajectory t in S^{ω} is a *good* trajectory. Alternatively, we can see a goal as a fuzzy set of trajectories (see [Ruspini, 1991b] for a semantic interpretation of multivalued logics and fuzzy sets in terms of utility and desirability).

Specifying goals as (fuzzy) sets of full trajectories is customary in control theory [Bellman, 1961], but is less common in AI approaches to planning (but see [Georgeff, 1987]). One advantage of using (18) is that both goals involving the *achievement* and goals involving the *maintenance* of some condition can be expressed in the same form. More precisely, let P be a (multivalued) predicate; we define the following goals:

$$\text{ACHIEVE}[P](t) = \sup_{s \in t} \mathcal{T}(P,s) \tag{19}$$

$$\text{MAINTAIN}[P](t) = \inf_{s \in t} \mathcal{T}(P,s) \tag{20}$$

$$\text{ACHIEVE!}[P](t) = \sup_{i:s_i \in t} \inf_{i \leq j \leq n} \mathcal{T}(P,s_j) \tag{21}$$

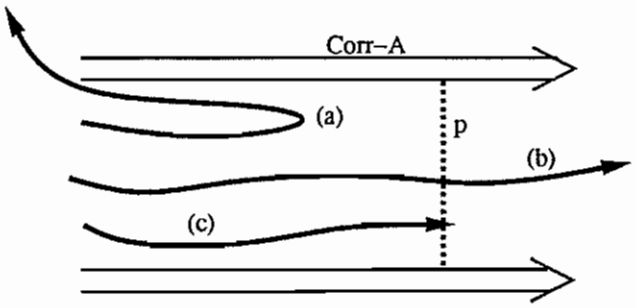


Figure 11: Trajectories and goals. (a) satisfies ACHIEVE[at(Corr-A)], but not MAINTAIN[at(Corr-A)] or ACHIEVE![at(Corr-A)], because it leaves the corridor. (b) satisfies MAINTAIN[at(Corr-A)] and ACHIEVE[$at(p)$], but not ACHIEVE![$at(p)$]. (c) satisfies all of MAINTAIN[at(Corr-A)], ACHIEVE[$at(p)$], and ACHIEVE![$at(p)$].

where $t = (s_0, s_1, \ldots, s_n)$ is any trajectory in S^ω. Intuitively, ACHIEVE[P] is the goal of making P as true as possible: 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: the goodness of a trajectory according to MAINTAIN[P] is the minimum truth value of P along the whole trajectory. And ACHIEVE![P](t) is a stronger form of achievement, where P is required to be eventually true, and to stay true until the end of the trajectory. Figure 11 gives examples of trajectories that satisfy or falsify different goals. For the Follow CS, traveling along a corridor to a position p, typical goals would be:

MAINTAIN[at(corr)]
MAINTAIN[$\neg near(side(\text{corr}))$]
MAINTAIN[$oriented$(corr)]
ACHIEVE![$at(p)$]

That is, the agent stays near the center of the corridor, oriented along its direction, and arrives at the position p and stays there.

There are several measures for characterizing goals and their relation to behaviors. The simplest is the notion of goal consistency: any goal should be satisfiable in some environment.

(22) $$Cons_G(G) = \sup_{t \in S^\omega} G(t)$$

This measure will become important when we consider the mutual consistency of two different goals.

The next measure embodies our main step to relating behaviors and goals: a measure of how well a behavior achieves a goal. Informally, the CS B is a "good" behavior with

respect to a goal G if every trajectory satisfying B makes G true. This informal definition has a natural translation into multivalued logic operators.

DEFINITION 4.1 (GOOD BEHAVIORS)

Let $B = (C, D, O)$ be a control structure, and G a goal. The goodness *of B with respect to G, denoted by $Good(B, G)$, is defined by:*

$$Good(B, G) = \inf_{t \in \mathrm{fin}(S^\omega, O)} (G(t) \oslash (Adm_D(t) \circledast C(t))) .$$

We shall often say that B is a *behavior for* G to mean informally that $Good(B, G)$ is "reasonably" close to 1.

There are several parts of this definition that need further explanation. Recall that a trajectory is a set of internal states, which, given the perceptual embedding of the agent, reflect external conditions. The set of trajectories $\mathrm{fin}(S^\omega, O)$ are those that contain the artifact O, and which are "long enough." The reason short trajectories cannot be considered is that they may not give the behavior enough time to accomplish G. For example, if the behavior is to reach a goal point distant from the agent, then a one-step trajectory will obviously fail to satisfy the goal, although it may be admissible for the behavior. What constitutes "long enough" depends on the behavior B. Technically, we can always consider trajectories with some arbitrarily large number of steps.

The subtle part of this definition lies in the interaction of the contextual conditions of B with the goal predicate. When part of a trajectory is "out of context," any control will be admissible from B's point of view, and the trajectory can vary arbitrarily. In the definition of *Good*, the context condition is conjoined with admissibility, so that out-of-context trajectories need not satisfy the goal. This leads to interpreting $Good(B, G)$ as: for every admissible trajectory t, either t satisfies G, or it is out of context for B.[7] This is fair: we should not expect a behavior to work when its contextual conditions are unsatisfied. Unfortunately, this means that even if $Good(B, G)$ holds, there may never be a situation in which G is actually made true. Consider a behavior B designed to go through a doorway (Figure 12):

$$B = \langle \mathsf{near(door)}, \mathsf{Cross}, \mathsf{door} \rangle$$

The context is that the agent be in the vicinity of the door. $Good(B, \textsc{achieve}[at(\mathsf{door})])$ will be true if B moves the agent away from the door, since that brings the agent out

[7] It is interesting that this notion is similar to that of goal in Cohen and Levesque's theory of intention [Cohen and Levesque, 1990]: 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.

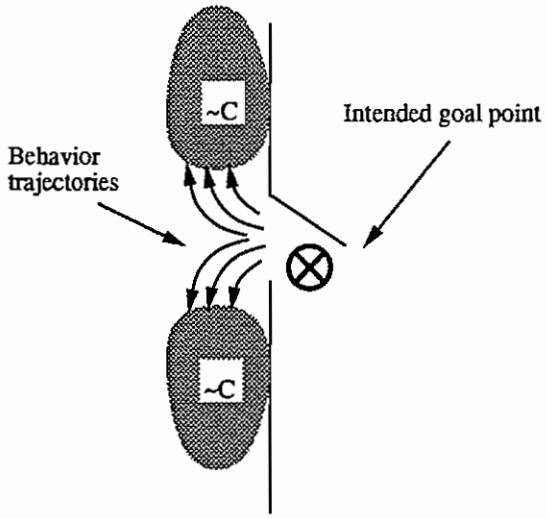


Figure 12: A good but unrealizable goal. The goal is to cross the doorway from left to right. The behavior produces trajectories which always go out of context, so the goal of crossing the doorway is never satisfied, although the behavior is *Good* with respect to the goal.

of context for the behavior. Clearly *Good*-ness is not strong enough a relation between behaviors and goals. It must also be the case that the behavior satisfies the goal for some trajectory. We call this condition *realizibility*.

DEFINITION 4.2 (REALIZABLE BEHAVIORS)

Let $B = \langle C, D, O \rangle$ *be a control structure, and* G *a goal.* B realizing G *is defined by*

$$Realizable(B, G) = \sup_{\iota \in \mathrm{fin}(S^\omega, O)} (G(t) \circledast Adm_D(t) \circledast C(t)) .$$

Obviously, any realizable goal will also be consistent. An admissible, in-context trajectory satisfying G is called a *realizable trajectory* of B for G. For a behavior to reliably achieve a goal, it should be *Good* with respect to the goal, and also realize it in the largest possible set of environments.

It will be convenient to also have a weaker form of *Good*ness, one where a behavior may take some time to stabilize before it satisfies the goal. We say that a behavior is *eventually good* for a goal if any trajectory that it can possibly produce has some state s_0 such that the s_0 on that trajectory is satisfactory for the goal. Formally,

DEFINITION 4.3 (EVENTUALLY GOOD BEHAVIORS)

Let $B = \langle C, D, O \rangle$ be a control structure, and G a goal. The eventual goodness *of B with respect to G, denoted by $Eventually(B, G)$, is defined by:*

$$Eventually(B, G) = \inf_{t \in \mathrm{fin}(S^\omega, O)} \sup_{t' \in \mathrm{tail}(t)} \left(G(t') \oslash \left(Adm_D(t') \circledast C(t') \right) \right),$$

where $\mathrm{tail}(t)$ *denotes the set of sub-sequences of t that have the same end state as t.*

Notice that the notions of *Eventually* good and *Good* are equivalent for goals of type ACHIEVE and ACHIEVE!.

In our practical experience with writing behaviors to satisfy goals, we have found some interesting challenges. The first is computational. Since a goal is a function over trajectories, in many cases one must examine a large number of trajectories in order to produce behavior that satisfies a goal for an arbitrary environment. This is true, for example, in the navigation domain: to implement a Go-To-Control-Point behavior for an arbitrary geometric arrangement of obstacles, one must employ potentially expensive path-planning methods to find an admissible trajectory to the goal point. Besides being expensive, this method violates the spirit of the control schema approach. Instead, we would like to perform a simple local calculation to generate plausible moves based on current conditions. Of course, behaviors formed using only local information may not be admissible for a goal; but in a large number of environments they may perform acceptably. Most of the behaviors we write use very simple local rules; for example, the Go-To-Control-Point behavior does not compute trajectories to the goal point, but relies on simple computations based on orientation and distance from the goal to choose a control.

A second experience we have had is that writing behaviors to satisfy complex goals is difficult. Generally speaking, we try to write behaviors that satisfy a single goal in a small set of environments, because it is easy to do. Then, by writing auxiliary behaviors, we can expand the applicable environments and add additional goals. This compositional methodology makes the life of the behavior designer much easier (it should also make it easier to learn new behaviors automatically). To carry through the compositional methodology, we need to show how behaviors and their goals combine.

4.1 Combining Behaviors

Combination and interaction of several behaviors occurs routinely in the activity of situated agents. Combining behaviors can be a way of extending the context of applicability of some

of them, like when I use my walking to the table as a way of extending my ability to grasp a glass to the entire room, or it can be a way of achieving several goals simultaneously, like when I eat a biscuit while reading this paper. In order to account for the latter case, we need to make the notion of simultaneous achievement of goals precise.

As a simple example of goal conjunction, consider the case when a robot wants to reach a certain location l_1 while avoiding another location l_2. Given any trajectory t, we can write the degree of satisfaction of each goal as:

$$\begin{aligned} G_1 \equiv \quad & \text{ACHIEVE}[\text{at}(l_1)](t) &=& \sup_{s \in t} \mathcal{T}(\text{at}(l_1), s) \\ G_2 \equiv \quad & \text{MAINTAIN}[\neg\text{at}(l_2)](t) &=& \inf_{s \in t} (1 - \mathcal{T}(\text{at}(l_2), s)) \end{aligned}$$

The degree by which a trajectory t satisfies both constraints G_1 and G_2 depends on how much t "passes close" to l_1 at some point, and on how much it is uniformly not close to l_2. In general, we use the $\circledast$ (in our case, min) operation to compute the above "and":

DEFINITION 4.4 (GOAL CONJUNCTION) *Let G_1 and G_2 be two goals. The* conjunction *of G_1 and G_2, denoted by $G_1 \otimes G_2$, is the goal defined, for any trajectory t in S^ω, by:*

$$(G_1 \otimes G_2)(t) = G_1(t) \circledast G_2(t) .$$

Conjunction of goals is a way to trade off several goals into one set of trajectories that best satisfy all of them. There are two important caveats here. First, the characteristics of this tradeoff depend on the particular triangular norm $\circledast$ employed. By using the min, we are saying that in order for the conjunction to be satisfactory at the level α, each of the conjuncts must be satisfied at least at the level α. An alternative choice for the $\circledast$ operator would be the product: this would give us a different notion of tradeoff, where a decrease in the level of satisfaction of one conjunct can be traded off for a corresponding increase in the satisfaction of the other one. Many other choices for $\circledast$ can be made, leading to trade-offs of different types (e.g., [Klir and Folger, 1988]); one interesting possibility, that we do not explore here, would be to allow the simultaneous use of different operators for different combinations of goals.

The second caveat regards the consistency of the combined goal, and the corresponding meaning of the combination. If two goals are not compatible, their conjunction loses its interest: promoting the conjunction will not significantly promote any of the conjuncts independently. For instance, suppose that in the example above l_1 and l_2 are very close, that is, for any s, if $\mathcal{T}(\text{at}(l_1), s)$ is high, so is $\mathcal{T}(\text{at}(l_2), s)$, and *vice versa.* Then, no trajectory can

be very satisfactory for both goals simultaneously, and any way to achieve the conjunction $G_1 \otimes G_2$ would still behave poorly with respect to each of G_1 and G_2 individually. We quantify the *mutual consistency* of two goals G_1 and G_2 by the following measure, derived from the consistency measure given in (22) above.

$$Cons_G[G_1, G_2] = \frac{Cons_G(\, G_1 \otimes G_2\,)}{Cons_G(G_1) \circledast Cons_G(G_2)} \tag{23}$$

It is immediate to verify that $0 \leq Cons_G[G_1, G_2] \leq 1$ (and is undefined if $Cons_G(G_1) = 0$ or $Cons_G(G_1) = 0$).

4.2 Blending Behaviors

We are now ready to show how to compose, or *blend* control structures to form complex behaviors, and to study the properties of these behaviors. We base our blending on one single pattern to compose desirability functions: the context-depending blending given in Def. 2.5. Depending on how the contexts are used in the combination, we can distinguish three different flavors of blending, which results in behaviors with different properties.

DEFINITION 4.5 (BLENDING OF CONTROL STRUCTURES)
Let $B_1 = \langle C_1, D_1, O_1 \rangle$ and $B_2 = \langle C_2, D_2, O_2 \rangle$ be two control structures. We define the following control structures:

$$\begin{aligned}
\textsc{disj}(B_1, B_2) &= \langle\, (C_1 \cup C_2), (D_1^{\downarrow C_1} \otimes D_2^{\downarrow C_2}), (O_1 \cup O_2)\,\rangle \\
\textsc{conj}(B_1, B_2) &= \langle\, (C_1 \cap C_2), (D_1^{\downarrow C_1 \cap C_2} \otimes D_2^{\downarrow C_1 \cap C_2}), (O_1 \cup O_2)\,\rangle \\
\textsc{chain}(B_1, B_2) &= \langle\, (C_1 \cup C_2), (D_1^{\downarrow C_1 \setminus C_2} \otimes D_2^{\downarrow C_2}), (O_1 \cup O_2)\,\rangle\,.
\end{aligned}$$

The disjunctive operator DISJ provides the simplest form of combination: it builds a control structure that considers each one of the original desirability functions in its context, conjunctive combination CONJ builds a more focused control structure that considers both original desirability functions in a common context, and the chaining operator CHAIN builds a control structure that extends the context of the second desirability function by using the first desirability when the second one does not apply.

Every combination of control structures can be expressed in terms of disjunctive blending alone, provided that the right contexts are considered. More precisely, we have:

$$\textsc{conj}(B_1, B_2) = \textsc{disj}(\langle (C_1 \cap C_2), D_1, O_1 \rangle, \langle (C_1 \cap C_2), D_2, O_2 \rangle) \tag{24}$$

$$\textsc{chain}(B_1, B_2) = \textsc{disj}(\langle (C_1 \setminus C_2), D_1, O_1 \rangle, \langle C_2, D_2, O_2 \rangle)\,. \tag{25}$$

Hence, disjunction of behaviors can be used as a canonical form for expressing composed behaviors. Moreover, as DISJ is associative and commutative, we simply write $\text{DISJ}(B_1, \ldots, B_n)$ to mean the disjunction of behaviors $B_1, \ldots, B_n$ taken in any order. Notice that the context of this disjunction is $(C_1 \cup \ldots \cup C_n)$, where C_i is the context of $B_i, i = 1, \ldots, n$.

Composition of control structures is our strategy to build complex behaviors. Later in the paper we will show that this composition produces good results in practice, but first we want to formally analyze the result of composing control structures by the above operators. We need to relate the properties of the blended control structure to those of the two component ones. In particular, given that B_1 is a behavior for some goal G_1 and B_2 is a behavior for some goal G_2, we want to see whether the blending (of any sort) of G_1 and G_2 is a behavior for a goal related in some way to G_1 and G_2. It turns out that this is indeed the case. The exact relation between the goals satisfied by the original behaviors, and the one satisfied by the blended behavior, depends on which type of combination is used. The following composition theorems characterize our basic types of combination.

THEOREM 4.1 (CONJUNCTION OF BEHAVIORS) *Let $B_1 = \langle C_1, D_1, O_1 \rangle$ and $B_2 = \langle C_2, D_2, O_2 \rangle$ be two control structures, and let G_1, G_2 be two goals. Then, if $Good(B_1, G_1) \geq \alpha$ and $Good(B_2, G_2) \geq \beta$, then*

$$Good[(\text{CONJ}(B_1, B_2)), (G_1 \otimes G_2)] \geq \alpha \circledast \beta .$$

Theorem 4.1 tells us that we can build behaviors that address more goals simultaneously by conjuncting 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. However, it is important to notice that Theorem 4.1 says only that the composed behavior is good for the conjoint goal: as we have seen above, promoting the conjoint goal may or may not be a satisfactory way to promote each of the two conjuncts, depending on how compatible they are. It is easy to verify that the degree to which any behavior B can be simultaneously good for both G_1 and G_2 cannot exceed the mutual consistency $Cons_G[G_1, G_2]$, no matter how good B is for the conjunction $\text{CONJ}(G_1, G_2)$. That is:

PROPOSITION 4.2 *Let G_1, G_2 be two goals, and B be any control structure. Then*

$$Good(B, G_1) \circledast Good(B, G_2) \leq Cons_G(G_1, G_2) .$$

The practical advice here is that care should be taken when conjunctively combining behaviors that their mutual consistency be reasonably high.

Another important point that Theorem 4.1 is not guaranteed by is the *Realizable*-ability of the conjunctive goal by the conjunction of behavior. For example, consider a room, Room-1, with two doors, Door-1 and Door-2, and consider two behaviors, B_1 and B_2, for entering Room-1 through Door-1 and through Door-2, respectively. Each behavior is individually *Good* for its own goal, hence, the conjunct behavior $\text{CONJ}(B_1, B_2)$ is *Good* for the conjunction of the two goals: entering Room-1. Moreover, this conjunction is trivially consistent. Unfortunately, there is no trajectory that is admissible for both B_1 and B_2 — the term $Adm_D(t)$ in Def. 4.1 and Def. 4.2 is identically zero, hence $\mathit{Realizable}(\text{CONJ}(B_1, B_2), G_1 \otimes G_2) = 0$. One should be careful that the desirability functions of the behaviors being combined be mutually consistent, that is, there are trajectories in the common context $C_1 \cap C_2$ that are admissible for both behaviors. In general, we measure the mutual consistency of two behaviors B_1 and B_2 by the following function:

$$Cons_{\text{B}}(B_1, B_2) = \frac{\sup_{t \in S^\omega} (C_1(t) \circledast C_2(t) \circledast Adm_{D_1^{\downarrow C_1} \otimes D_2^{\downarrow C_2}}(t))}{\sup_{t \in S^\omega} (C_1(t) \circledast C_2(t))} \tag{26}$$

where we let $Cons_{\text{B}}(B_1, B_2) = 1$ if the denominator is zero. Hence, to ensure that the combined behavior be *Realizable*, we need to make sure that the mutual consistency of the two components is "reasonably" close to 1.[8]

Figure 13 illustrates the conjunction of two behaviors B_1 and B_2, that are *Good* for G_1 and G_2 in the context C_1 and C_2, respectively. Here, the goals are the ACHIEVEment of one of the states in the rectangles. The thick borders indicate the context and goal of the conjoint behavior. Trajectories (b) and (c) are admissible for both B_1 and B_2, and hence are admissible for the conjunction; they satisfy the conjoint goal. Trajectory (a) is also admissible, but is out of the conjunctive context, and does not count for *Goodness*. Trajectory (d) is not admissible because it is not admissible for B_2.

Theorem 4.1 tells us that we can, with some proviso, conjoin behaviors to achieve more specific goals in a narrower context. We would like to prove a similar theorem for disjunctions of behavior: this would give us a way to enlarge the context of applicability to the union of the individual context. Unfortunately, very little can be said in general about the result of a behavior disjunction.

[8] Alternatively, the value of $Cons_{\text{B}}$ can be monitored during execution — for example, by checking whether we get into a situation where $D(s, a)$ is identically zero — and used to trigger some modification of the combined control. We shall say more about this strategy in Section 5.

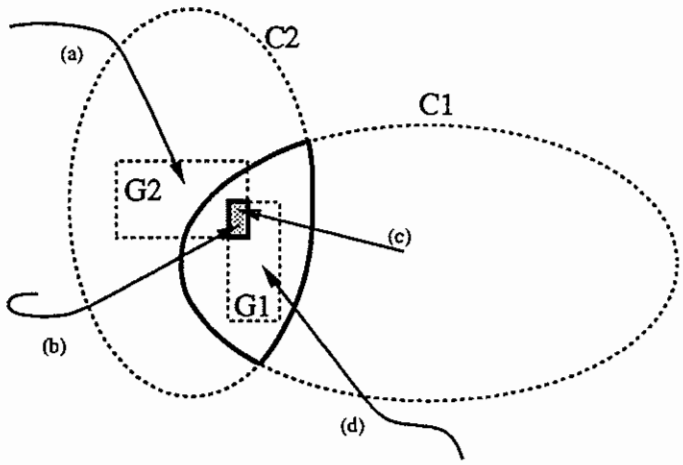


Figure 13: Conjunction of behaviors. The thick borders indicate the context and the goal for the conjunctive behavior. Trajectories (a), (b), and (c) are admissible for the conjunction.

THEOREM 4.3 (DISJUNCTION OF BEHAVIORS) *Let $B_1 = (C_1, D_1, O_1\rangle$ and $B_2 = \langle C_2, D_2, O_2\rangle$ be two control structures, and let G_1, G_2 be two goals. Then, if $Good(B_1, G_1) \geq \alpha$ and $Good(B_2, G_2) \geq \beta$, then*

$$Good[\text{DISJ}(B_1, B_2),\ G_1 \oplus G_2] \geq \alpha \circledast \beta\,.$$

where, for all t, $(G_1 \oplus G_2)(t) = G_1(t) \oplus G_2(t)$.

So, the only thing we can tell of $\text{DISJ}(B_1, B_2)$ is that it will be *Good* in the extended context $C_1 \cup C_2$ for one of G_1 or G_2 — but we do not know which one. In most cases, this information is too weak to be useful. However, there is one special case where we can extend the context and have a strong *Good*ness property: when one of the behaviors ACHIEVEs the context of the other one. In this case, the two behaviors cooperate to (eventually) satisfy the goal of the second behavior. Formally, we have the following:

THEOREM 4.4 (CHAINING OF BEHAVIORS)
Let $B_1 = \langle C_1, D_1, O_1\rangle$ and $B_2 = \langle C_2, D_2, O_2\rangle$ be two control structures, and let G be a goal. Then, if $Good(\, B_1, \text{ACHIEVE}[C_2]\,) \geq \alpha$ and $Eventually(B_2, G) \geq \beta$, then

$$Eventually(\,\text{CHAIN}(B_1, B_2),\, G\,) \geq \alpha \circledast \beta\,.$$

where $\text{ACHIEVE}(C_2)$ is the goal defined by (19) above.

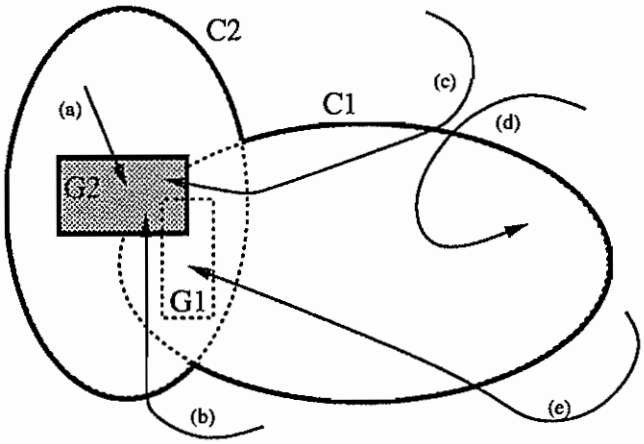


Figure 14: Chaining of behaviors. The thick borders indicate the context and the goal for the chained behavior. Trajectories (a), (b), and (c) are admissible for the chaining.

Theorem 4.4 tells us 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. More precisely, if the agent has a behavior B_2 to (eventually) satisfy a goal G under circumstances C_2, and if it has a behavior B_2 that, when applied under circumstances C_1, makes sure that circumstances C_2 hold, then the agent can build a new behavior that (eventually) satisfies G under the more general circumstances $C_2 \cup C_1$. This new behavior is obtained by considering the control suggestions of D_1 while in C_1 and not yet in C_2, and those of D_2 once in C_2. The consistency conditions required for the conjunctive combination to make sense are not so stringent in the case of chaining. On the one hand, mutual consistency of goals is not relevant, as chaining is not aimed at attaining both goals simultaneously, but only at attaining the first one as a step to the attainment of the second. On the other hand, and because the contexts of the two behaviors in the blending are disjoint, mutual behavior consistency is more easily the case — we only have to bother about the existence of a commonly agreed trajectory in the fuzzy transition zone from C_1 to C_2.

Figure 14 illustrates the result of CHAIN(B_1, B_2) with the same B_1 and B_2 used in Figure 13 above. Notice that the goal G_1 of B_1 is a subset of C_2; hence, B_1 is a *Good* behavior for ACHIEVEing C_2. The resulting behavior is *Good* for the goal G_2 in the context indicated by the thick borders. Every trajectory that enters $C_1 \cup C_2$ and is admissible for the chaining achieves G_2. In the picture, trajectories (a) and (b) are admissible for B_2, and lie entirely within C_2, and so are admissible for the chaining (notice that (b) would not be

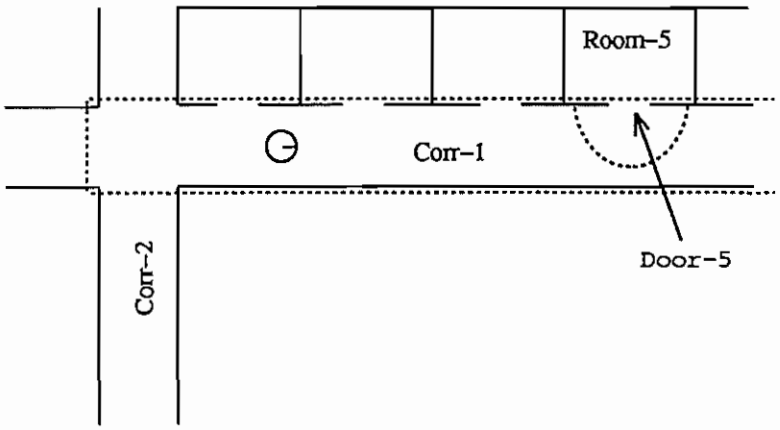


Figure 15: A robot in an office environment. The dashed areas indicate the contexts of the Follow and the Cross motor schemas.

admissible by B_1 alone). The effect of having extended the context is visible in (c) and (d): (d) is admissible for B_2 alone, but it is not admissible for the chaining, as it is not admissible for B_1 in its part in C_1; (c) is admissible for the chaining: it obeys B_1 when in $C_1 \setminus C_2$, and obeys B_2 when in C_2. (e) is admissible for B_1, but not for the chaining, as it does not obey B_2 in C_2.

The following example illustrates this point. Consider the situation in Figure 15, where a robot, sitting in corridor Corr-1, needs to reach room Room-5. The robot has a control schema, called Cross, that results in crossing a door if applied when the robot is near to that door. Hence, the following control structure is a good behavior (at some level, assumed close to 1) for reaching Room-5:

$$B_1 = \langle at(\text{Corr-1}) \wedge near(\text{Door-5}), \text{Cross}, \text{Door-5} \rangle$$

where Door-5 is the artifact used by Cross and 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 to ACHIEVE this context. The robot has a second control schema, Follow, that results in going down a corridor when applied inside that corridor. In particular, the Follow schema results in the robot eventually being near each object in Corr-1 (e.g., Door-5). Hence, the following control structure is a good behavior for the goal ACHIEVE[near(Door-5)] in our environment:[9]

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

[9]The artifact of the Follow schema indicates the direction to consider.

where Corr-1 is the artifact used by Follow and kept anchored to the actual corridor Corr-1. Theorem 4.4 ensures us that we can chain B_1 and B_2 to obtain a composite behavior $\textsc{chain}(B_2, B_1)$ that leads the robot into Room-5 in the larger context of being anywhere along Corr-1. More precisely, we have

$$Good(\,\textsc{chain}(B_2, B_1),\ \textsc{achieve}(\text{at(Room-5)})\,) \geq \alpha \circledast \beta\,,$$

where α and β are the goodness of B_1 and B_2 for their respective goals.

Notice that $\textsc{chain}(B_1, B_2)$ can be canonically expressed in a disjunctive form by using (25):

$$\begin{aligned}(27)\qquad &\textsc{disj}[\,\langle\ (\text{at(Corr-1)} \wedge \neg\text{near(Door-5)})\,,\ \text{Follow}\,,\ \text{Corr-1}\ \rangle\,],\\ &\qquad\langle\ \text{at(Corr-1)} \wedge \text{near(Door} - 5)\,,\ \text{Cross}\,,\ \text{Door-5}\ \rangle\,.\end{aligned}$$

4.3 Blending Reactive and Purposeful Behavior

We have illustrated the concepts of goals, control structures, and control schemas with examples from mobile robot navigation on our testbed, Flakey. Our experience in this domain has led us to formulate some guiding methodological principles for combining reactive and purposeful behaviors:

- Behaviors should be computationally inexpensive, being simple functions on local conditions.
- To make them easy to write, debug, and compose (and, in the future, 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 achieving single goals.
- Since the environment will contain uncertainty, complex behaviors should be reactive, rather than relying on precomputed trajectories.

This last principle is familiar from the recent literature on reactive planning, whose main motivation is to guarantee agents a high sensitivity and responsiveness to 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. Combining reactivity with goal-oriented behavior in a satisfactory way is a delicate task, as testified by the abundance of literature about the problem of obstacle avoidance during goal-oriented navigation in the mobile robot domain

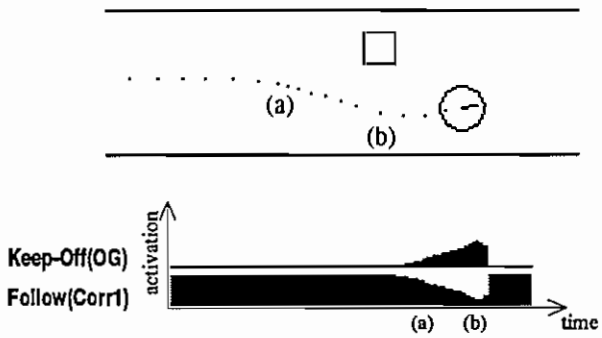


Figure 16: Blending corridor following with obstacle avoidance.

(Section 6.2). The principles cited above, implemented using the techniques of goal blending, have proven in our experience to be an exceptionally good method for real-world navigation tasks in uncertain environments.

To illustrate the concepts involved, we consider a simple example of blending behaviors to combine reactive and purposeful movement. We have already shown an example of this combination in Section 2 (Figure 7); here, we analyze that example more closely. Two control structures are chained together: one for following the corridor when there are no obstacles in the way, and one for avoiding any obstacle when it appears:

$$\text{CHAIN}[\langle\, \text{near(OG)},\ \text{Keep-Off},\ \text{OG}\,\rangle. \\ \langle\, (\text{at(Corr-1)} \wedge \neg\text{near(OG)}),\ \text{Follow},\ \text{Corr-1}\,\rangle] , \tag{28}$$

where Corr-1 is an artifact kept anchored to the corridor to follow, and OG stands for the obstacle-grid: a representation of the configuration of free space around the robot used to detect obstacles. In the case of our robot, Flakey, OG is built by collecting readings from the sonar and vision sensors into the Local Perceptual Space. For better convenience, we repeat in Figure 16 the run made by our robot, together with a plot of the level of activation of each control structure over time (i.e., the truth value of the corresponding context); control structures are named after their control schema and artifact. The plot shows how the Follow behavior gradually becomes "out of context" as the obstacle is increasingly *near*, and the context of Keep-Off becomes true: the weight given to the preferences expressed by the two control structures in the blend varies accordingly. Notice that the obstacle avoidance behavior does not completely take over the Follow one: the desirability expressed by the latter is still considered, with decreased weight, during the obstacle avoidance maneuvers. This is important when the robot has a choice between different ways to avoid an obstacle:

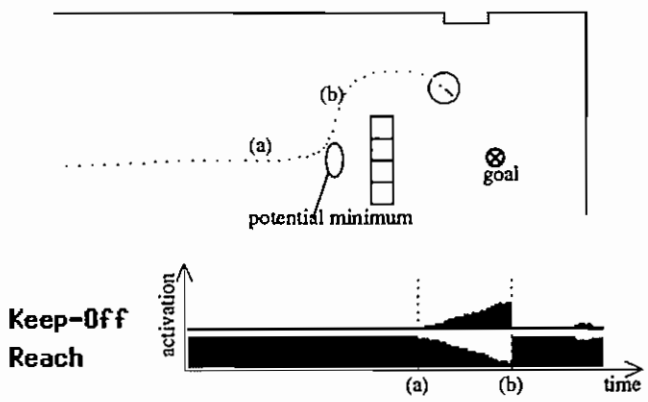


Figure 17: How context-dependent blending of behaviors helps avoiding a potential local minimum.

by considering the purposeful behavior, the robot chooses the avoidance strategy that is most compatible with the pursuing of its goal.

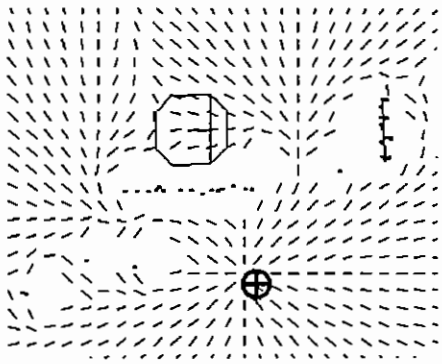
It is interesting to notice that the Follow and Keep-Off behaviors have been CHAINed together rather than CONJoined. As we noticed in Section 4, the Follow behavior (as most behaviors) is written to be a *Good* behavior for going down a corridor *provided that there are no physical impediments.* The Keep-Off behavior, on the other hand, is a good behavior for avoiding obstacles, that is, for bringing about a situation where there are no more physical impediments in the context where there are obstacles around. By CHAINing these behaviors, we can obtain the composite behavior illustrated above that is good for going down a corridor whether there are obstacles or not.

Local combination of behaviors, of some form or another, is widely cited in the robotic literature as a technique for mixing goal directedness and reactivity [Latombe, 1991]. One ubiquitous problem is the emergence of local minima (or maxima) in the combination. The problem is illustrated in Figure 17: the robot needs to mediate the tendency to move toward the goal, and the tendency to stay away from the obstacle. A straightforward combination of these two opposite tendencies may result in the production of a zone of local equilibrium (local minimum): when coming from the left edge, the robot would be first attracted and then trapped into this zone. The use of contexts, and of context-depending blending, provides a way around this problem is some situations. Figure 17 shows the path followed by Flakey in a simulated run, and the corresponding activation levels of the KEEP-OFF and REACH behaviors. In (a), Flakey has perceived the obstacle; as the obstacle becomes

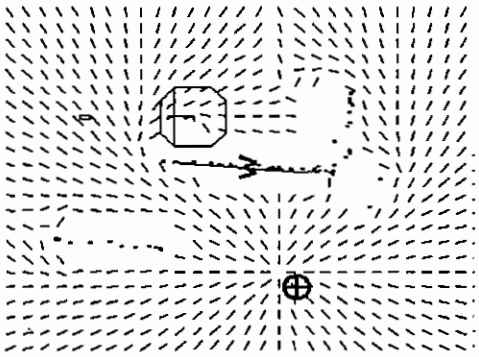
nearer, the Keep-Off behavior becomes more active, at the expenses of the Reach behavior. In this way, the "attractive power" of the goal is gradually shaded away by the obstacle, and Flakey responds more and more to the obstacle-avoidance suggestions alone. The Reach behavior regains importance, however, as soon as Flakey is out of danger (b).

While local minima and oscillatory behaviors can still emerge in the compositional behaviors presented here, in a large number of environments simple strategies generated by the context mechanism have proven effective. Global path-planning methods are better, however, when two conditions hold: (a) the obstacles are complex, and (b) sensing gives an accurate picture of this complexity. For the mobile robotics domain, neither of these conditions generally holds, and the local, compositional approach is more efficient than global path-planning methods. It also has the advantage of additivity of goals: if we want the robot to go slower or faster, or to take into account some other condition as it moves, we simply add a behavior to accomplish this goal in the relevant context.

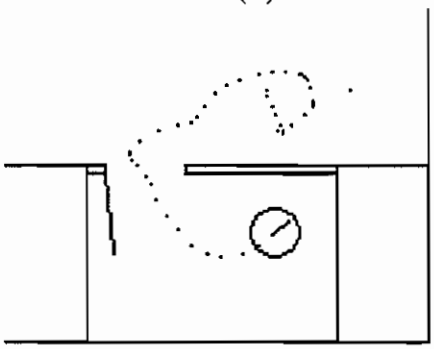
In some cases, however, there is no substitute for geometric reasoning to determine a satisfactory trajectory. In these cases, we have combined more expensive path-planning as global advice to local, reactive behaviors, similar to the methods of Payton [Payton *et al.*, 1990]. A gradient field is generated by a grid search from the goal point to the robot. At each point, the gradient gives the direction the robot should travel in the shortest path to the goal. Figure 18 shows a sample gradient generated when the robot is trying to enter a doorway. In the initial situation (a), the robot does not see all of the wall in front of it, and the gradient field leads around to the right. After moving to this area, the sensors find enough of the wall so that a new computation of the field shows the correct path. (c) shows the overall movement of the robot. The gradient method is useful for escaping from local minima that occur using simple reactive obstacle-avoidance strategies. However, it can be successfully employed only if enough perceptual information has been gathered to show all the obstacles between the robot and its goal.



(a)



(b)



(c)

Figure 18: Using a gradient method for path planning. (a) and (b) show the gradient field at two intermediate points. The dark circle is the position goal of the robot. (c) is the final robot motion.

5 Generating Complex Controllers

Reactivity to environmental conditions is an important property of agents. But an intelligent situated agent also 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 in the environment. A large part of the AI literature has been devoted to the problem of automatic generation of plans, and a number of solutions have been proposed, ranging from the off-line means-ends analysis of the "traditional" planning approach (e.g., [Wilkins, 1988]), to the real-time deliberation proposed by the advocates of the "reactive" approach (e.g., [Georgeff and Ingrand, 1989; Gat, 1992; Payton *et al.*, 1990; Rosenschein, 1987]).

In this section we show how the notion of *plan*, and automatic generation of plans, can be adapted to situated controllers. We start by noticing how possibly complex control structures can be seen as a particular type of plans, that we call *embedded*. We then proceed to show how control structures can be represented in a form that allows the agent to reason about them in order to generate embedded plans that achieve given goals. It is important to notice that the representation that we use is quite customary in terms of the representation and reasoning capabilities of current AI planners. This suggests that current AI approaches to plan generation can be used to generate embedded plans. This is in fact the case, and we report two experiments in the automatic generation of embedded plans: using a simple goal-regression planner, and using PRS-like[10] intention structures, respectively. In both these methods, the unique character of control structures bridges the gap between control of physical movement and more abstract characterizations of action. In addition, the use of multivalued logic allows us to define a number of measurable properties of embedded plans, and to use these measures to judge the *adequacy* of the plan to the current situation, and to engage in new deliberation when necessary. This last point is part of our current research, and we only hint at it briefly.

5.1 Complex Behaviors as Embedded Plans

A number of authors have claimed that plans to be executed by agents situated in the real world should be expressed in some declarative form, such as the "*situation* $\rightarrow$ *action*" rules used in many reactive systems (e.g., [Drummond, 1989]). Further, it has been suggested that this declarative representation should be considered as a source of information, or resource,

[10]PRS is the Procedural Reasoning System of Georgeff and Ingrand [Georgeff and Ingrand, 1989].

to *inform* rather than to *control* the activity of the agent [Suchman, 1987]. According to this view, the agent's activity is the result of a continuous and only partially conscious interaction between the agent and the environment, and the role of a plan is to give guidelines to orient this activity toward the achievement of intended goals. The view of plans as resources for actions has been incorporated by Payton [Payton *et al.*, 1990] in his *internalized plans.*

This view of plans encompasses control structures — in particular, the complex structures obtained by blended composition. To see this, recall that a CS provides the agent with a preferential criterium to choose between different possible control actions in any situation. The actual movements performed, as well as the sequence of activation of the different control schemas involved, are not specified in the CS, but depend on the interaction between the robot and its environment. For example, consider again the combined control structure (27) used in the last section to navigate to Room-5. In a typical execution, the robot will initially be influenced by the Follow schema alone to get to Door-5; when the robot is near to Door-5, Follow will blend with Cross to induce the robot to orient toward the door, and finally to cross the door. However, there is nothing in (27) that commits execution to this sequencing: if King Kong, passing by, were to push our small robot in front of Door-5, execution would directly start by considering the Cross schema. All that (27) gives is advice on the preferred move in any situation: the fact that (27) is a *good behavior* for the goal of getting to Room-5 guarantees that local advice is coherent with the attainment of the final goal.

To make our use of complex behaviors as plans explicit, we introduce the notion of an *embedded plan.* To have an embedded plan for a goal is, for us, to have a complex control that, under well-specified conditions, results in the production of movements that are expected to satisfy that goal. As complex control structures can always be expressed as disjunctive combinations ((24) and (25)), we define embedded plans in terms of disjunctive sets of control structures:

DEFINITION 5.1 (EMBEDDED PLANS)

Let $\mathcal{P} = \{B_1, B_2, \ldots, B_n\}$ *be any set of control structures, and* G *any goal. We call* $\mathcal{P}$ *an* embedded plan, *and define the* goodness *of* $\mathcal{P}$ *with respect to* G *as:*

$$Good(\mathcal{P}, G) = Good(\text{DISJ}(B_1, B_2, \ldots, B_n), G)\,.$$

For simplicity, we may simply say that $\mathcal{P}$ is an embedded plan for G, meaning that $Good(\mathcal{P}, G)$ is "reasonably" close to 1. For example, as the combined CS in (27) is a behavior for achieving the goal of being in Room-5 in the environment in Figure 15, then

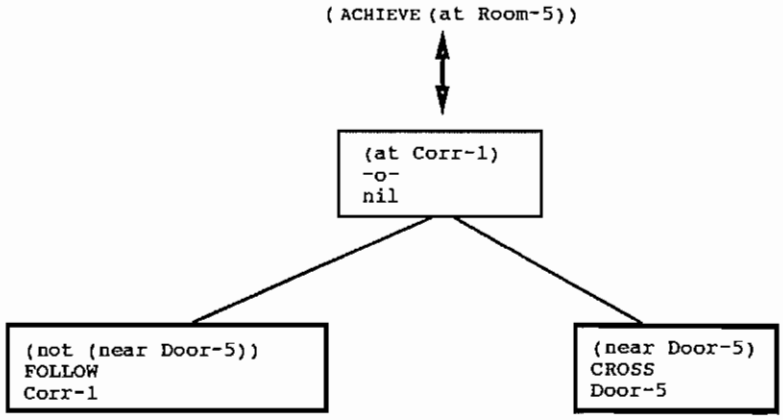


Figure 19: Graphical representation of an embedded plan.

the following set of control structures

$$\left\{ \begin{array}{l} \langle\, (\mathsf{at}(\text{Corr-1}) \wedge \neg \mathsf{near}(\text{Door-5}))\,,\ \mathsf{Follow},\ \text{Corr-1}\,\rangle\,], \\ \langle\, (\mathsf{at}(\text{Corr-1}) \wedge \mathsf{near}(\text{Door-5}))\,,\ \mathsf{Cross},\ \text{Door-5}\,\rangle \end{array} \right\} \tag{29}$$

is an embedded plan for the goal

$$\textsc{achieve}[\,\mathrm{at}(\text{Room-5})\,]\,.$$

To execute an embedded plan $\{B_1, B_2, \ldots, B_n\}$ we need to compute the desirability function of $\textsc{disj}(B_1, B_2, \ldots, B_n)$, and to use it to decide, at each state, a control to apply. Having the plans represented in the canonical disjunctive forms guarantees that this desirability function is always in the form of a generalized context-depending blending (cf. 2.5):

$$D_1^{\downarrow C_1} \otimes D_2^{\downarrow C_2} \otimes \ldots \otimes D_n^{\downarrow C_n}\,.$$

This allows us to use one single mechanism for execution: in particular, the implementation suggestions discussed in Section 2.3 can be used for executing any embedded plan (this is the way execution proceeds in Flakey).

The embedded plan (29) can be represented in a graphical form as shown in Figure 19. The boxes with thick borders represent the control structures that constitute the plan *strictu senso*. Although these control structures are sufficient to fully characterize the plan, we may include boxes corresponding to intermediate nodes that represent the disjunctive blending of their children: in our simple plan there is only one such box, drawn with thin borders,

corresponding to the full plan. In more complex plans, there may be many intermediate nodes: which intermediate nodes are represented depends on, among other things, the way the plan is generated — a hierarchical planner may include intermediate nodes for any level of abstraction. Intermediate nodes do not list the artifacts and the desirability functions, as they are immediately derived from their children. However, they include a context: each intermediate node stores the common (conjunctive) factor of the contexts of all its children. This way of distributing the context is a convenient device to speed up computation and to help monitor the execution of the plan (see below). In this way the context in each node directly shows when the subplan rooted at that node is active — a useful indication in complex plans.

One prerequisite to being able to automatically generate embedded plans for an agent is to have a representation of the basic behaviors available to that agent — that is, a specification of the control schemas for which the agent has a "built-in" desirability function (written as control rules), together with a description of the class of goals for which each schema can produce a good behavior and the corresponding applicability conditions. We represent abstract behaviors by means of *templates*: for example, the following is the template used by Flakey for the Follow behavior:

```
Template: FOLLOW
            With-binding: corridor(?c), part-of(?c, ?p)
           Preconditions: at(?c)
                  Effect: near(?p)
                Behavior: Follow
                Artifact: ?c
```

The "With-Binding" field constrains the instantiation of the variables in the template for a given environment; the Preconditions specify the context under which applying the Behavior control schema to the Artifact is expected to result in the Effect being true. Hence, this template tells us that any control structure of the form

$$B_1 = \langle\, \mathsf{at}(C), \mathsf{Follow}, C \,\rangle$$

will be a good behavior for the goal

$$\textsc{achieve}[\mathsf{at}(P)]$$

in any environment where C is anchored to a corridor and P to a location that is in that corridor.[11] One important problem is how can we be sure that the family of control

[11] A more careful representation would explicitly indicate the measure of the goodness of a family of control structures for the class of goals expressed in the template.

structures defined by a template is actually a good behavior for the corresponding goal under the specified conditions. In our experiments with Flakey we took a very pragmatic approach, and relied on the programmer who wrote the control schema to "guarantee" that this is the case. We are investigating techniques for automatically generating sets of fuzzy rules that provably promote a given goal [Ruspini and Saffiotti, 1993].

5.2 Goal-regression Planning

A simple way to use templates to generate embedded plans is to use a classical goal regression strategy [Nilsson, 1980]. As an illustration, consider once again the problem of going to Room-5 given the environment in Figure 15: the plan (29) above could be generated by using a simple goal-regression strategy from the goal "at(Room-5)" given the FOLLOW template above and the following one:

```
Template: CROSS
            With-binding: door(?d), leads-to(?d, ?p)
           Preconditions: near(?d)
                  Effect: at(?p)
                Behavior: Cross
                Artifact: ?d
```

Notice that the context of each behavior in a chain so generated has to be modified by subtracting the context of the next behavior (which is, normally, the "Effect" of the behavior itself). Theorem 4.4 provides a justification for this goal-regression procedure.

To verify the feasibility of automatic generation of embedded plans, we have implemented a simple goal-regression planner for Flakey.[12] This planner uses templates like the ones above; however, when considering the actual behaviors available to Flakey, some subtleties emerge. For example, Flakey's Cross behavior is more restrictive than the one assumed above: it works properly only when the robot is approximately facing the door to cross. This is expressed in the CROSS template by restricting the "Precondition" to: *near(?d)* $\wedge$ *facing(?d)*. A second behavior, Face, can achieve *facing(?d)* when used from near door *?d*. A goal regression planner will typically CHAIN Face with Cross to obtain a behavior to cross a door in the extended context where Flakey is near that door. Notice that in some executions the Face behavior may never become active — for example, if Flakey is already heading to the door when approaching it. In other executions, Face may

[12]This planner is extremely simple and limited: for instance, it cannot deal with conjunctive goals. We are currently performing experiments where we use SIPE [Wilkins, 1988] to generate embedded plans that solve more complex tasks.

be needed a second time after having already been successfully executed — for example, if Flakey is forced to drastically change its orientation to avoid an obstacle close to the door while executing the Cross behavior. These common cases of unforeseen events, sometimes referred to as "protection violations," are dealt with effortlessly by the context-depending blending mechanism.

Another interesting case of interacting behaviors is related to the need of Follow to keep its artifact anchored to the corridor's walls. We have seen in Section 3 how Follow relies on a Sense behavior to help the anchoring process. Here we give the actual templates used by Flakey for the Follow and the Sense behaviors:

```
Template: FOLLOW
             With-binding: corridor(?c), part-of(?c, ?p)
            Preconditions: at(?c)
                 Maintain: anchored(?c)
                   Effect: near(?p)
                 Behavior: Follow
                 Artifact: ?c

Template: SENSE
             With-binding:  corridor(?c)
            Preconditions:  at(?c)
                   Effect:  anchored(?c)
                 Behavior:  Sense
                 Artifact:  ?c
```

The Maintain precondition is not used for chaining, but for starting a parallel behavior that guarantees that the required condition holds during all the activation of the Follow behavior. The Sense behavior, as discussed above, biases execution towards slow movements and little turning, as this is known to help sonar-based perception of walls; to perceive a corridor, Flakey needs to perceive its walls.

Given these templates, the planner would generate, for our previous example for going to Room-5, the following two behaviors in place of the first one in (29):

$$\langle\, (\mathsf{at(Corr\text{-}1)} \wedge \neg\mathsf{near(Door\text{-}5)} \wedge \neg\mathsf{anchored(Corr\text{-}1)})\,,\ \mathsf{Sense}\,,\ \mathsf{Corr\text{-}1}\,\rangle\,,$$
$$\langle\, (\mathsf{at(Corr\text{-}1)} \wedge \neg\mathsf{near(Door\text{-}5)})\,,\ \mathsf{Follow}\,,\ \mathsf{Corr\text{-}1}\,\rangle\,.$$

Figure 20 shows the graphical representation of this embedded subplan. Notice the context of the Sense behavior: when the corridor is already anchored, Sense is not needed, and the robot can proceed at normal cruising speed.

Figure 21 shows the graph of Flakey's embedded plan to reach Room-5 in the environ-

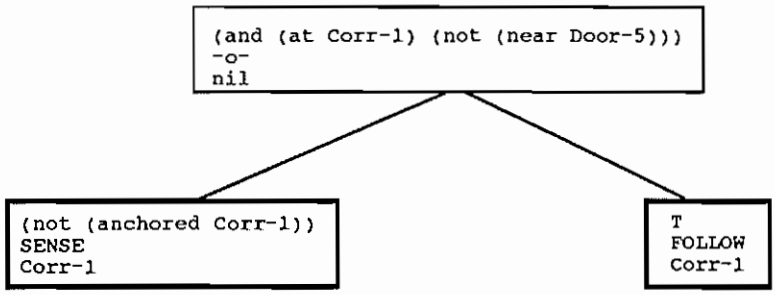


Figure 20: A subplan to follow a corridor and keep it anchored.

ment sketched in Figure 15.[13] The context of this plan is that of Flakey being anywhere in Corr-1, Corr-2, or Room-5. The reader will easily recognize all the combination patterns discussed above in this plan. Notice in particular that the Keep-Off behavior has been chained to extend the applicability of the plan to situations where there are obstacles around. Also recall our convention for factoring the context out of the children: the actual context of a control structure B is given by ANDing the context in its node with those in all its ancestors. For example, the full context of the leftmost Sense is:

```
(and (not (near OG)) (not (at R5)) (not (near D5))
     (at C2) (not (at C1)) (not (registered C2))).
```

The arrows and numbers in the picture illustrate the computation of the context for one specific situation (the one marked *(g)* in Figure 22 below): an initial unitary value is propagated down the tree, and ANDed (through min) at each node with the truth value of the expression in that node.

Figure 22 shows an instance of an execution of this plan, along with the corresponding level of activation over time of each control structure in the plan. Notice the emerging sequencing of behaviors at (d), and around (g), and the interaction between purposeful behaviors and reactive obstacle avoidance — after (c), and around (d), (e) and (g). The interaction right after (g) is worth a comment. The Cross behavior relies on prior information about the door position; as this turns out to be fairly off, Keep-Off raises to avoid colliding with the edge of the wall. The combined effect is that Flakey engages the opening that is "more or less at the estimated position." The ability to integrate map knowledge at

[13]This is one of the plans actually generated and executed by Flakey to navigate in the offices at SRI. The names of the artifacts are abbreviated by using its first letter and its number: e.g., C1 stands for Corr-1, the artifact kept anchored to the Corr-1 corridor.

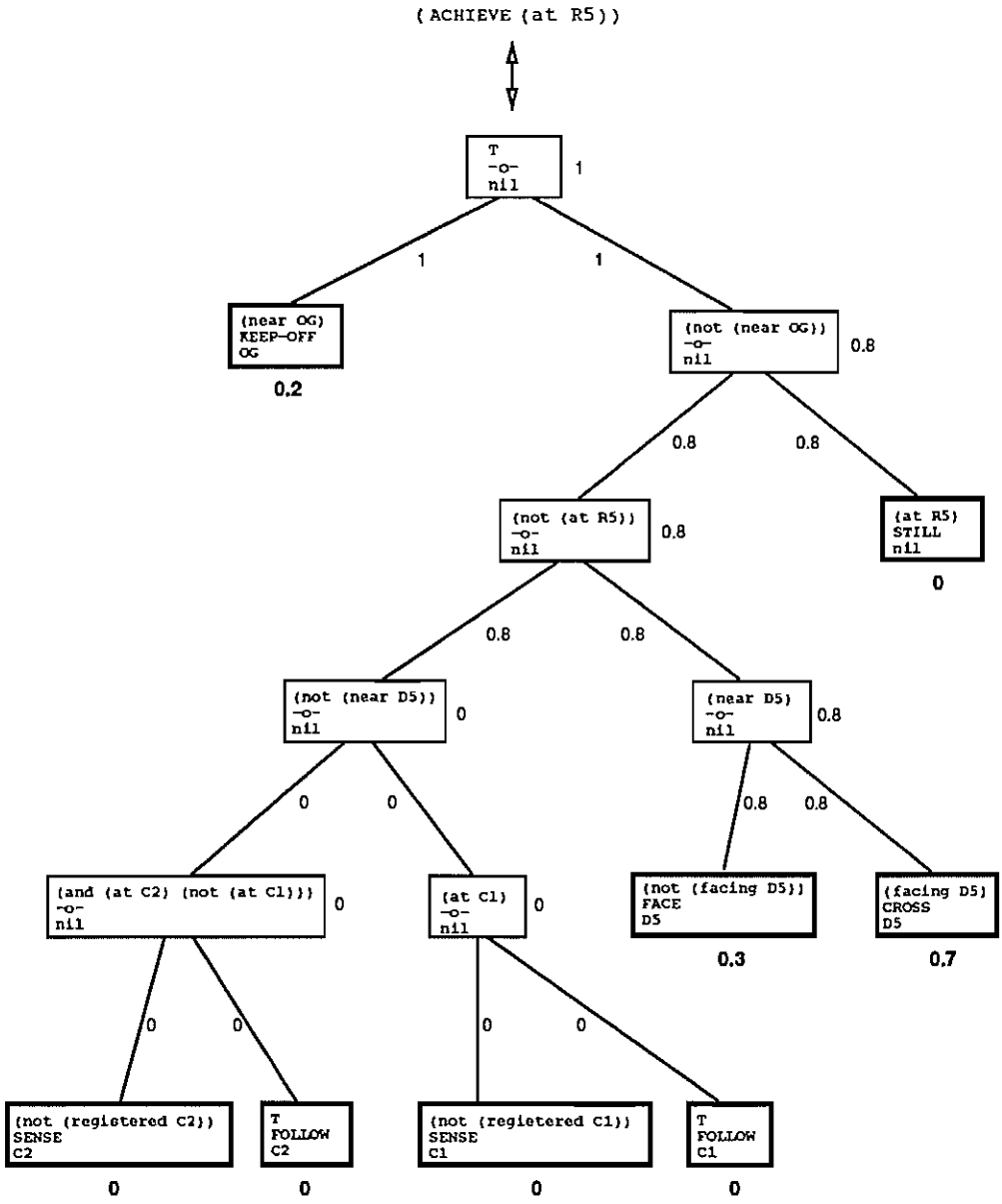


Figure 21: An embedded plan leading Flakey to Room-5. The numbers show the computation of the context for each control structure at a certain moment of execution (point (g) below).

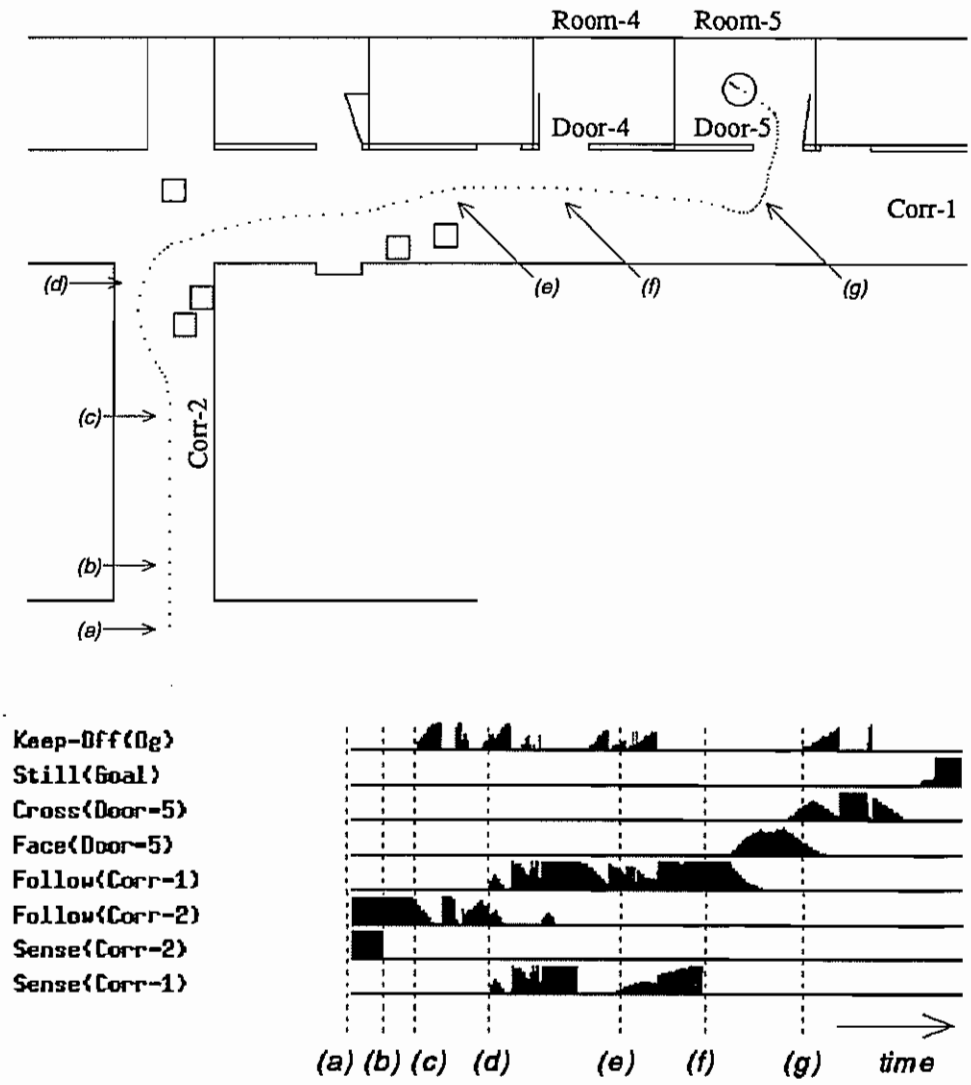


Figure 22: Sample execution of the embedded plan. The temporal evolution of CS activations is shown.

planning time with perceptual knowledge at run time is one advantage of using control structures as a representation of a plan shared between the symbolic level of the planner, and the perceptuo-motor level of the controller.

It is interesting to see how the Sense behaviors come into play during the execution. When entering a new corridor, in (a) and (d), Flakey needs to register it; after traveling a few feet, the walls are perceived, and Flakey can move more freely (b). However, if Flakey loses track of the walls for a while (e.g., a wall is occluded by boxes in (e)) the Sense behavior regains importance and slows down Flakey until reregistration occurs (f). Notice that no reasonable *a priori* order could have been established between the executions of corridor-following and corridor-sensing: the latter simply comes into play whenever Flakey is losing its anchor. Also, notice that explicit planning for perceptual actions could not be accounted for by geometric path-planning methods.

5.3 Monitoring

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 common technique to deal with this problem, adopted in AI since the Shakey era [Nilsson, 1984], consists of enriching the execution apparatus with *monitors* whose task is to detect deviations from the assumptions made at planning time, and to intervene accordingly. The basic idea is to produce idealized plans that completely abstract from the messy aspects of real execution, and then add some mechanisms to take care of exceptional situations when "things do not proceed as planned."

Embedded plans do not escape the risk of turning inadequate, and we discuss here some possible techniques for monitoring the execution of control structures. The meaning of monitoring, however, is different in our case from the one in the case of classical plans. Embedded plans are not step-by-step procedures to be executed, but specifications of the relative utility of various control actions in each situation (inside a given context) from the point of view of satisfying a given goal: embedded plans are better thought of as sources of information than as programs. In particular, there is no "expected" course of execution to be monitored, but only situations for which the information in an embedded plan is *relevant*, and situations for which it is not — in the latter case, the plan will not be adequate.[14] The task of the monitor is to detect whether or not the current plan $\mathcal{P}$ is adequate for the satisfaction of the current goal G in the current situation s; more precisely, the following

[14] A more complex case is when this information is *wrong*, as in the case in which the description of a behavior in a template does not correctly reflect that behavior's characteristics. In this paper, we limit ourselves to the simpler case.

conditions need to be true:

1. The context of the embedded plan is satisfied, i.e., $C(s)$ is "reasonably" close to 1;
2. The plan is good for the current goal, i.e., $Good(\mathcal{P}, G)$ is "reasonably" close to 1.

Our most elementary form of monitoring consists of observing the values of $C(s)$ and $Good(\mathcal{P}, G)$, and engaging in a repair process when these values become unsatisfactory.

To be more specific, let us consider an agent who is always executing some embedded plan and has always a goal. By default, this will be an Idle behavior aimed at keeping the agent in some stable idle state. This behavior is simple in the case of Flakey (i.e., do nothing), but may be fairly complex in the case of an underwater robot (e.g., keep a stable and stationary position in a collision-safe area). Suppose then that the agent is given a new goal:[15] for example, suppose that Flakey, idle in its lower left corner in Figure 15, is given the goal to go to Room-5. The Idle plan is not a *Good* plan for this goal, and Flakey is left without any applicable suggestion as to the control actions to execute. A monitor process testing the value of $Good(\mathcal{P}, G)$ should detect that condition 2 above is not satisfied, and respond appropriately: in a typical case, Flakey may ask the services of its goal-regression planner to generate a new plan that is *Good* for the new goal.

A second case where a plan may become inadequate is when it is "out of context". When the plan is out of context, it is not useful, and the agent needs to find a way to enlarge the plan's context to cover the current situation. For example, suppose that Flakey, executing the navigation plan in Figure 15, enters instead Room-4 while avoiding an unfortunate configuration of obstacles, or a stationary group of people. No control structure in the plan is applicable to the context of Flakey being *at(Room-4)*: the plan is out of context. A monitor checking condition 1 above can now come into play and modify the current plan. To this respect, the structure of the plan and the evaluation of the contexts can provide some useful hints. Figure 23 shows the value of the context for each node in the plan when Flakey is *at(Room-4)*: notice that the context value for each leaf node is 0. Also notice that the value of the context of the node marked by '*', $\neg$*near(Door-5)*, is 1, but the context of all its children is 0 — node '*' is left "floating." This suggests two alternative strategies to modifying the plan to make it cover the current situation. The first possibility is to find a way to ACHIEVE the context of some of the children of '*' from the present situation. In many cases this can be done relatively easily: in our case, the action that brought Flakey out of context can be easily reversed by a Face behavior and a Cross behavior on Door-4).

[15] Or generate a new goal by itself, e.g., because it is getting hungry. Where do the goals come from is an important question that we are not addressing in this paper.

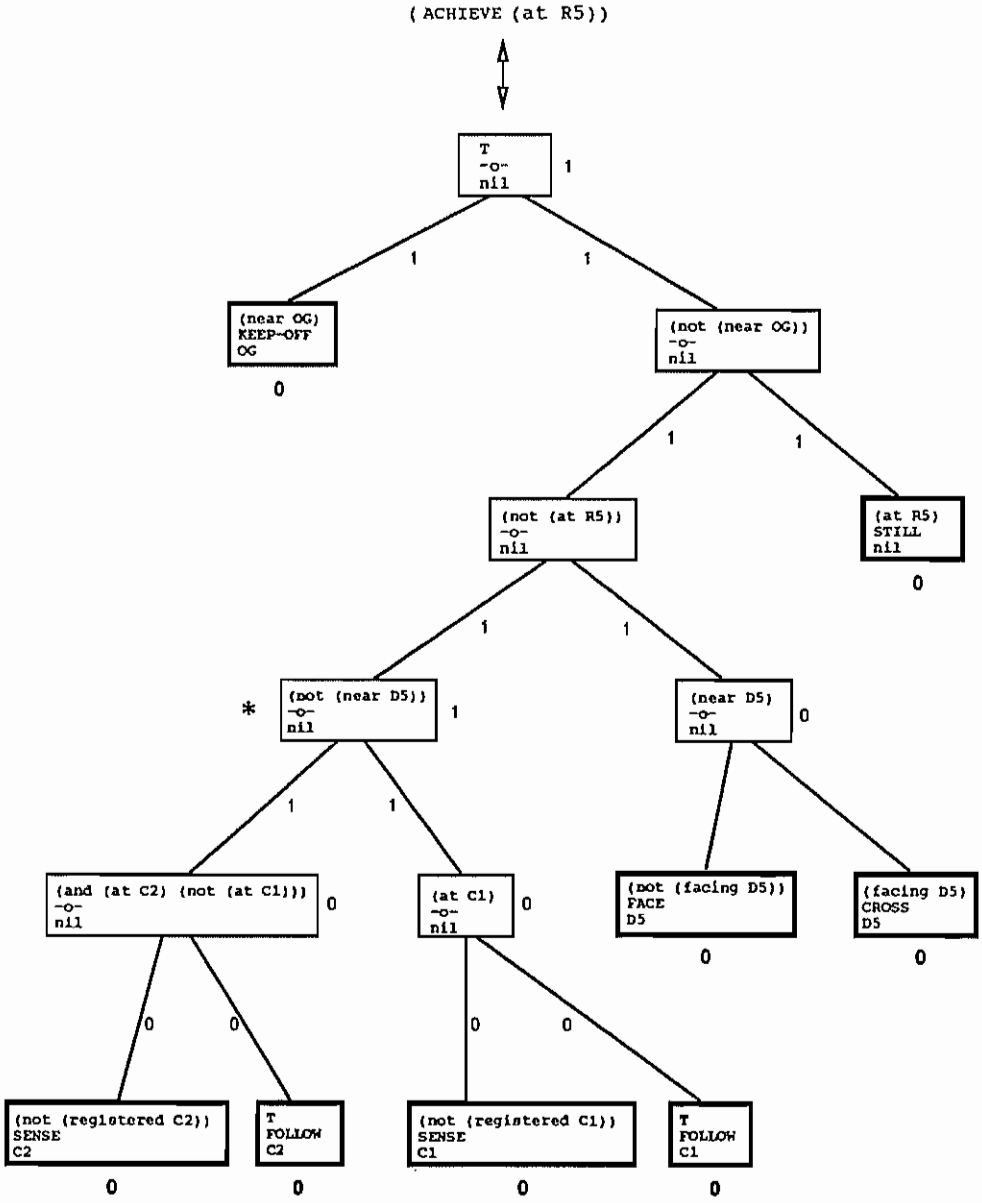


Figure 23: Values of the contexts in the situation where Flakey mistakenly entered Room-4. The star marks the most specific active node in the plan.

The second possibility is to find an alternative way to achieve, from the present situation, the condition that the '*' node is meant to achieve: *near(Door-5)*.[16] Flakey may invoke the services of the goal-regression planner to find a way to achieve this goal from the present situation. Whether the plan is extended to ACHIEVE the context of the more specific active node ('*'), or that of its children, the result is a new plan that is as *Good* as the original one for the present goal G, and is in context. Unfortunately, there is no guarantee in general that the extended plan will be the "best" plan for satisfying G.

We can define other useful measures for a plan — for example, a measure of how much the artifacts being currently used are *anchored*; or a measure of how much the trajectory actually produced so far by the controller satisfy the goal (i.e., how *effective* is the current execution of the plan). These and other measures can be used to monitor the execution of a plan, and to decide when some modification of the plan is needed. The purpose of the brief exposition above, however, is mainly to illustrate a general strategy that can be used to interleave execution and deliberation in embedded plans. This important issue is a subject of our current research.

5.4 Run-time Deliberation

A complex control, whether hand-coded or automatically generated by a planner, cannot account for all contingencies, and we have seen above how to recognize a situation where an embedded plan is no more adequate. Reactive planners such as Firby's RAP [Firby, 1989] or Georgeff's PRS [Georgeff and Ingrand, 1989] introduce a limited amount of deliberation into the execution process. This type of strategic planning can make the execution of higher-level actions more robust, especially where the state of the world is uncertain at planning time. One can think of run-time deliberation as tactical planning: limited deliberation using rehearsed procedures for coping with contingencies.

We have experimented with run-time decision-making using intention schemas similar to those of PRS, adapting them to the control structure templates defined in this section. An intention schema consists of a goal, a precondition, a trigger condition, and a body.

Goal The intended effect of the schema, expressed as a predication with parameters, e.g., $at(?P)$.

Precondition A necessary condition for the instantiation of the schema, e.g., $near(?P)$.

[16] Notice that this would require that we enrich our representation of intermediate nodes in the plan to include the intended goal of each node.

Trigger A condition which, if it becomes true, will cause the schema to be invoked. Trigger conditions are a facility for interrupting current goal execution to respond to events.

Body A finite state machine encoding a sequence of steps the schema will execute.

An intention schema is a strategy for accomplishing its goal; it performs a limited amount of deliberation, and can invoke subintentions and CSs to accomplish its task. Typically, it will initiate a behavior to gather more information, make a decision based on that information, and initiate another behavior to achieve its goal. It can also monitor the progress of initiated behaviors, and switch to another one when they are not making progress.

Control structure templates fit well into this methodology. They have the form of intention schemas, but do not have a trigger condition. Typically, templates are invoked by higher-level intention schemas as a way of performing actions to satisfy a goal.

The basic PRS cycle is illustrated in Figure 24. A goal stack contains the current set of goals to be achieved or maintained, together with their executing schemata. The goal stack is like the currently executing path in a plan tree. At the bottom of the stack is the highest-level goal and its intention body; at the top is a set of control structures that are executing to fulfill goals lower in the stack. On every cycle, the current state of the world is updated and checked. Completed intentions are removed from the stack, the bodies of the remaining intentions are processed, and any intention schemas whose triggering conditions are satisfied are placed on the stack.

To test limited deliberation for making decisions during obstacle avoidance, we implemented two strategies. These strategies interleave actions that acquire more information with actions for movement towards a goal. In the first, we used a simple method for deciding which way to go around a corridor obstacle. On first contact with an object that projects far into the corridor, the sensors provide only limited information about the best way to go around. 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. If the obstacle is directly in front, the information-gathering strategy is chosen. If the obstacle is to one side, then the immediate movement strategy is chosen. The intention schemas work quite well on simple obstacles in the cor-

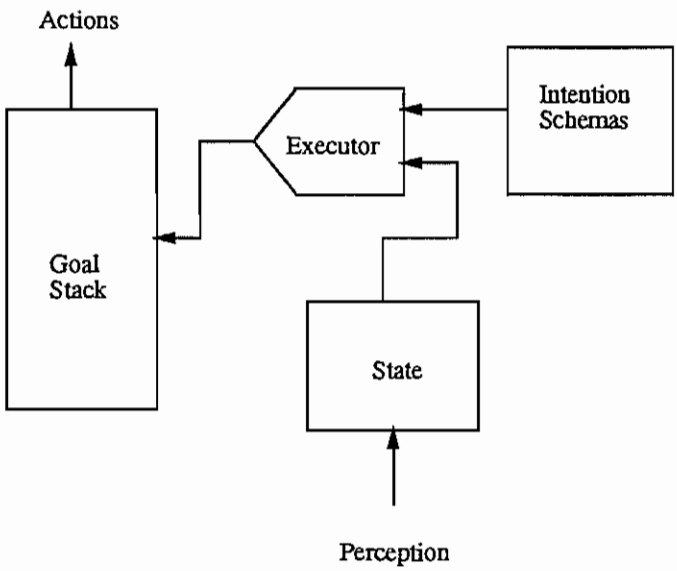


Figure 24: A simple PRS architecture. Intention schemas are invoked by an executive when their triggering conditions become true, or when they are needed to satisfy a goal on the stack.

ridor. Figure 25 shows an example of the immediate movement strategy being invoked, in (a), the wrong decision is made initially, then corrected as more of the obstacle comes into view; (b) shows an extended run.

Like embedded plans, intention schemas are another way of sequencing behaviors based on the current situation. Unlike embedded plans, the behaviors are not coordinated by means of additional context, but rather by the actions of the intention schemas. In addition, not all contingencies must be planned for ahead of time; 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, and deciding when to locate landmarks needed for self-localization. The trigger facility is especially helpful in coping with contingencies that must interrupt the current task.

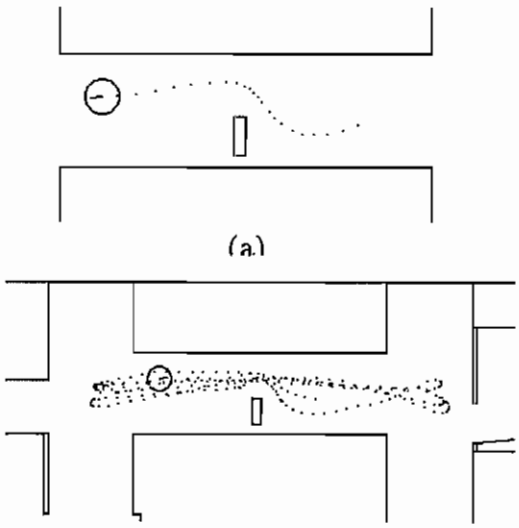


Figure 25: Two runs of the obstacle-avoidance strategy. In (b), Flakey was patrolling the corridor up and down.

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 approach to complex controllers.

The first subsection discusses the two methodologies of subsumption architectures and situated automata in relation to complex controllers. The second gives an overview of reactive planning architectures, which have generated considerable interest as the most promising method for intelligent control of mobile robots. Then we discuss plan compilation techniques that are similar to those presented in Section 5. Finally, we look at technical issues involved in alternative approaches to low-level control: potential fields and circuit methods.

6.1 Methodologies

Both the situated automata of Rosenschein and Kaelbling [Rosenschein, 1987; Kaelbling and Rosenschein, 1990] and the subsumption architecture of Brooks and his students [Brooks, 1982; Connell, 1990] are methodologies for producing embedded agents that perform complex tasks. In part we have borrowed from these in developing the complex control structure 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.

6.1.1 Situated automata

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. The general form of the automata is a finite state machine consisting of an update function and an action function with bounded execution times [Kaelbling, 1988]. On every cycle, the update function takes sensor input and updates the internal state, and then the action function produces an output vector for physical movement.

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, that is, 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. Consider the design of a simple embedded agent to patrol an area. It may be that, because of environmental constraints and the sensor suite of the agent, detecting the presence of the intruder is a simple matter of looking for a certain combination of sensor readings. There is no need to model the geometry of the patrolled area, or the very complicated physical structure of people. Rather, the agent can have a few bits of internal state representing presence or absence of an intruder, and its perceptual machinery, the update function, can be a simple function on the sensors.

In a similar manner, agents constructed using situated automata theory need not have complicated planners and executors to produce action. For constrained environments and simple tasks, decisions about which physical movement to make could be made very quickly as functions from the internal state. Again, the patrol robot serves as an example. If it is to patrol a corridor, and its sensors can reliably signal the ends and sides of the corridor, then its action function could cause direct physical movement by a simple computation using the sensor readings. By avoiding a commitment to analogical representations, and the consequent heavy computational investment in perceptual and planning processes, agents created using situated automata methodology can be computationally cheap, efficient, and reactive.

It should be emphasized that situated automata theory is not incompatible with the traditional planning approach, or our theory of complex controllers; 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 well developed enough to satisfy such an ambitious program. Its main practical success has been a suite of development tools: REX, to produce bounded-depth combinatorial circuits from low-level descriptions; and GAPPS and RULER [Kaelbling, 1988] for generating circuit descriptions from more abstract goals. GAPPS has been used to generate controllers for mobile robot navigation; we discuss it below.

6.1.2 Subsumption architectures

Subsumption architectures have many points in common with situated automata theory, but without the formal emphasis of the latter. It is an agent-constructing methodology that is oriented towards task decomposition. Each task is accomplished by a behavior, which accomplishes sensing, computation, and acting. Subsumption architectures are even stronger than situated automata theory in that they reject 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. In more restrictive versions of the architecture pursued by Connell [Connell, 1990], even internal state is eliminated, so that behaviors become simple transfer functions from sensor inputs to control outputs.

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. The idea is that if the higher levels cannot provide guidance, lower levels still cause the agent to act reasonably, for example, not bump into obstacles. 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 extremely influential in the mobile robotics 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 control structures interact with the LPS: more reactive behaviors can access raw sensor readings, while more purposeful behaviors use more complex perceptual routines. In fact, the very notion of control structures themselves, with artifacts representing perceptual information, and control schemas implementing control actions, is in the spirit of subsumption architecture, since each control structure is oriented towards accomplishing a particular goal. But the complex control approach differs 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 control structures whose purpose is to facilitate perceptual processes of recognition and anchoring. Of course, the response of the subsumption architecture approach might very well be that such processes are superfluous. There is room for debate here, and certainly the subsumption approach has achieved impressive results, with Connell's can-retrieving robot [Connell, 1990] and Mataric's navigation experiments [Mataric, 1990]. We believe that, while many

of the ideas of the subsumption architecture are valid, the approach will not be able to cope with more complex tasks and environments unless it makes contact with deliberation processes and analogical representations of the environment.

A second feature of complex control is the use of multivalued logic to formalize the behavior of the controller. Subsumption architectures must offer some way of putting behaviors together, since the results of higher-level ones must be combined with those of lower levels. Usually, this is in the form of a suppression mechanism, whose inspiration comes from studies of neurological systems of simple animals. These methods may be adequate as a programming technique, but it is difficult to prove any properties about them, or to accomplish the kinds of sophisticated goal trade-offs 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 26. 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 subfunctions 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 usual unit of representation is the *task*, and the second layer incorporates an interpreter and execution monitor for tasks. 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 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, for example, a task failing or completing.

There are many different instantiations of this architecture, including SSS [Connell, 1992], ATLANTIS [Gat, 1992], TCA [Simmons, 1990], RAPs [Firby, 1989; Firby, 1987; McDermott, 1990a], AuRA [Arkin, 1990b], and the reactive planners of [Payton, 1990; Payton *et al.*, 1990]. Most of these concentrate on the interaction between the top two layers, developing sequencers and integrating them with planning technology. Our approach

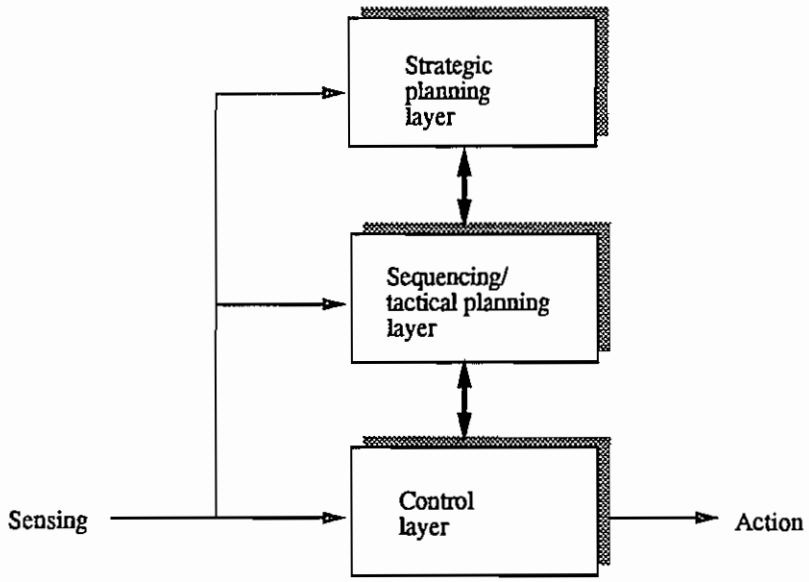


Figure 26: A typical hybrid architecture for reactive planners (after Connell's SSS system). The bottom layer is a controller that implements behaviors. The sequencer or tactical planning layer is in charge of initiating and monitoring behaviors. The strategic layer deliberates about larger courses of action, and sends executable plans to the tactical layer.

borrows heavily from the hybrid architecture, and we are comfortable with the notion of higher levels of abstraction providing guidance for lower control levels. We have concentrated on two important and not fully developed aspects of the hybrid planning framework. One is the relation between the control and sequencing layers. We have used the technology of multivalued logic to provide a principled way of composing complex controllers, thus easing the burden of the sequencing layer. We have also developed the embedding properties necessary to tie the control/sequencing level to the more abstract planning level.

6.3 Compiling Plans

One of the features of the complex controller methodology we presented is that new controllers can be synthesized by compilation of abstract goals. The synthesis is made possible by control structures, which bridge the gap between control schemas as descriptions of movement and action operators as descriptions of preconditions and effects. Synthesis is a convenient property, because it automates the process of producing complex controllers: all that is necessary is to define a suite of primitive behaviors as control structures. Standard planning technology can then be used to generate a plan, the leaves of which are control structures that achieve the goal.

Synthesizing controllers by compiling plans is not new to this work. We have been influenced by several other systems that compile plans into executable structures. These include the early triangle tables [Nilsson, 1985] and their later reincarnation as teleo-reactive nets [Nilsson, 1992], the GAPPS compiler [Kaelbling, 1990], and universal plans [Schoppers, 1987].

6.3.1 GAPPS

GAPPS [Kaelbling, 1988] is a language for specifying control programs as goal-reduction rules. The language allows specification of primitive control actions such as turning or moving forward, along with reduction rules that specify how abstract goals are to be decomposed. Goals can be given as maintaining or achieving a predicate, closed under boolean combination and some special constructs such as prioritization. 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. The goal reductions are detailed procedural directions for creating these rules; they cannot, for example, do deliberation about future courses of events. Later development of GAPPS incorporated goal-regression compilation using operator descriptions [Kaelbling, 1990], which gives GAPPS a more abstract way of representing

action, and enables it to do predictive planning.

Like the present work, GAPPS can be viewed as a method of synthesizing complex controllers. Our two synthesis methods, goal-regression planning and run-time deliberation, are similar to GAPPS goal-regression and goal-reduction methods, respectively: goal regression allows more general predictive planning, while run-time deliberation uses the goal-reduction rules of PRS. However, the PRS intention schemas are invoked at run time; in future work, we might benefit by explicitly compiling them into sets of control structures for frequently invoked goals.

Where the two systems differ is in the level of detail of complex action specification. We have concentrated on the formal and practical aspects of how control schemas combine to produce complex behavior. GAPPS control actions are simple effector commands, and conjoining controls generated by two different goals is possible only 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.

We have also explored the embedding of control schemas using the perceptual component and artifacts, and the consequences for reactive and purposeful movement. Again, this is not incompatible with GAPPS; GAPPS simply assumed the right perceptual predicates were available.

6.3.2 Teleo-reactive trees

When, by the end of the sixties, the developers of the STRIPS planning system had to cope with the difficulties of having Shakey, the robot, physically execute the produced plans in a "real" world, they devised a number of mechanisms that, in some form or another, are still around in many of the existing robot architectures [Nilsson, 1984; Fikes and Nilsson, 1971]. One of these was a representation for plans, called "triangle tables", that helped in monitoring the execution of the plan, and deciding which steps to perform and when. The basic idea behind the triangle table was to represent, together with each action in the plan, the condition under which that action is to be activated. While executing a strictly sequential plan is like executing a program — a program counter points at the action to be executed, and is incremented after the action completes — executing a triangle table plan is best compared to the evaluation of a set of production rules: all the conditions for each action in the plan are evaluated, in a given order, and the first action whose conditions are true in the present situation is executed. When the action completes, a new cycle begins.

The internal organization of a triangle table simplifies the evaluation of the conditions, and provides directionality — each action is expected to produce the conditions for another one that is closer to the goal. Although it is only recently that the idea of plans as sets of *condition* → *action* rules has been clearly spelled out and extensively developed [Suchman, 1987; Schoppers, 1987; Drummond, 1989; Payton *et al.*, 1990], it appears that more than the seeds of this idea were present in Shakey.

Most of the agent architectures that adhere to the view of plans as *condition* → *action* rules put these rules in the real-time control loop that describes the agent's activity. This implies that the evaluation of the conditions should be performed in bounded (and short!) time, and that the actions should be elementary action steps. This is not the case with triangle tables: there, the conditions are arbitrarily complex predicates with free variables to be instantiated, and actions can be as complex as pushing a box all the way to a given location. Moved by these considerations, Nilsson has recently developed a new formalism for plans, called *teleo-reactive trees* (TR-trees), that combines the intuition of triangle tables with the idea of continuous feedback [Nilsson, 1992]. Like triangle tables, TR-trees can be described as sets of *condition* → *action* rules, and require that the action of each rule (normally) produce the condition for the activation of a rule "closer" to the goal. Unlike triangle tables, TR-trees are continuously evaluated inside a real-time control loop.[17] At every tick of the agent control clock, all the preconditions (propositions whose truth can be computed in bounded time) are evaluated, starting from the rule "closest" to the goal, and the first action with a true precondition is activated; activation is an asynchronous process, and the control loop does not have to wait for the action to complete. At the next tick, the loop will repeat.

Nilsson emphasizes the fixed-cycle control character of his TR-trees by giving them a "circuit semantics": a TR-tree is described as a digital circuit that computes control outputs from sensor inputs. So, TR-trees can be seen as an abstract formalism to represent complex controllers. In this respect, TR-trees are very similar to, and have partially inspired, the complex controllers, or embedded plans, defined in Section 5. In fact, a TR-tree can be seen as a special case of embedded plan where: contexts are categorical (i.e., $C(s) \in \{0,1\}$); desirability functions are single valued and categorical; and only CHAINing is used to combine behaviors. That is, TR-trees do not allow parallelism in the execution of actions,[18] and

[17] There are more differences between TR-trees and triangle tables, including the fact that TR-trees can represent disjunctive paths to a goal, that they can be composed into a hierarchy, and that the TR-tree representation seems to be more amenable to manipulation, e.g., for learning [Nilsson, 1992].

[18] Nilsson and his students are working to extend the formalism to parallel goals and actions [Benson, 1993].

do not allow any sort of weighting. Hence, the main difference between TR-trees and our embedded plans is that TR-trees cannot capture the context-depending blending of behaviors that is the basis of our behavior composition technique. TR-trees have not been extensively applied to real robots: we believe that, when they are, some form of weighted arbitration similar to our multivalued logic-based mechanism will be needed.

A second aspect that links our complex controllers to TR-trees is that both can be easily generated by using conventional planning techniques. In both cases, the key to this is that the activation conditions of the rules (i.e., our contexts) are expressed in a logical form, and that the basic composition operators (CHAIN and, in our case, CONJ) have a direct correspondent in the typical goal-regression strategy. At the present state of development, our complex controllers have an advantage over TR-trees in that our behavior composition is formally justified.

6.3.3 Universal plans

The idea of representing plans as *situation* $\rightarrow$ *action* rules comports a different perspective on the role of a plan and on the task of a planner. On the one hand, this perspective substitutes the notion of plans as programs to be executed by one of plans as specification of reactions to situations; on the other hand, this interpretation entails a view of planning as the process of generating the correct reaction rules from the point of view of eventually achieving a given goal. The notion of "initial state" has disappeared (or, at least, it is secondary) in this interpretation.

The above arguments were first brought to light by Schoppers [Schoppers, 1987] in an attempt to provide a notion of plan adequate to a robot's execution in a noisy, uncertain and dynamic environment. In Schoppers's proposal, a plan is created only with respect to a goal state, disregarding the initial situation. The task of the planner is to partition (part of) the set of possible states according to the reaction that the agent should produce in that situation from the viewpoint of achieving the goal. One immediate question is how many possible states do we want to consider. Schoppers proposes an extreme solution where we consider *all* the situations that can possibly emerge given our set of states (but we still have to choose the granularity of the states). Hence, the name of *universal plans* for these plans.

It is interesting to notice that Schoppers's approach lies on the borderline between planning and control: a universal plan can be seen as a feedback controller providing an input/output mapping specific to the achievement of the given goal. Universal plans, however, were not originally conceived in this light: in particular, they were not clearly stated

over a restricted set of input variables, and did not touch the problem of the bounded control cycle time. Maybe, if Schoppers had described his universal plans as particular forms of controllers, some of the initial critics about the computational intractability of universal plans would have not been moved.

If we accept the interpretion of universal plans as a form of automatically generated controllers, our embedded plans can be seen as a more control-oriented version of universal plans. More important, the idea of "extending the context" that drives our generation of complex controllers is similar to the notion of "partitioning the space of possible situations" that Schoppers uses to generate his universal plans. Like Schoppers, we generate an embedded plan by starting from the goal, and a behavior that can achieve this goal in a limited context, and CHAINing more behaviors to it to extend the applicability of the plan to larger and larger contexts. It is natural to ask how much should we enlarge this context. The most conservative solution, the one we have adopted in Section 5, is to stop as soon as the context includes the current situation — that is, as soon as we have a plan that can lead us to the goal from the current state. The other extreme is, of course, to enlarge the context until it possibly covers all the states; this would produce a universal plan. But the suggestion of redundancy at the heart of universal plans is probably more usefully implemented by intermediate solutions: ones where we let the context cover some extra situations that are likely to occur. This would give us a precompiled answer to some not-so-unforeseen exceptions, and be a convenient alternative to extending the context to cover exceptions only as they arise at running time.

Finally, another plan formalism based on *situation* $\rightarrow$ *action* rules has been proposed by Drummond [Drummond, 1989]. Drummond takes the idea of plans as sources of information seriously, and defines the notion of *Situated Control Rules* (SCR) as the basic constituent of a plan. In his architecture, SCRs are given to an execution module to be treated as *constraints* that limit the behavior produced by the executor. Hence, Drummond's plans only help, or inform, an execution that can occur independently. Moreover, the inherent modularity of SCRs simplifies incremental synthesis and dynamic modification of plans. Finally, Drummond's architecture incorporates an interesting temporal projection mechanism that simulates possible future executions given the current plan, and detects undesired consequences at running time.

6.4 Theories of Control

An increasing number of researchers working on situated agents is looking at the integration of AI technique and techniques developed in the field of control theory (e.g., [Akar and

Umit, 1990; Dean and Wellman, 1991; Nilsson, 1992]). The hope, of course, is to be able to combine the high reactivity and mathematical properties of control systems with the high-level representation, reasoning, and planning capabilities of AI systems. The approach we have presented in this paper fully belongs to this category. In this subsection, we compare our approach with other control-oriented techniques: the potential field methods, and methods based on digital circuits.

6.4.1 Potential fields methods

Control schemas are similar in many respects to the so-called "artificial potential fields", first introduced by Khatib [Khatib, 1986] as a way to connect planning (path planning) and control (robot's path following), and now extensively used in the robotic domain [Latombe, 1991]. In the potential field approach, a goal is represented by a pseudo-force, which may be thought of as representatives of most desirable movements from that goal's viewpoint. For example, the goal of avoiding obstacles is represented by repulsive forces "emanating" from the obstacles; the goal of reaching a given location is represented by an attractive force at that location. These forces are then combined, as physical vectors, to produce a resultant force that summarizes their joint effect, and controls the robot's behavior. As pseudo-forces can in general be quickly computed and combined based on local perceptual information, potential field techniques can be highly reactive. The price to pay is the emergence of the shortcomings common to any approach based on local combination of behaviors — whether they are described by desirability measures, potential fields, motor schemas (see below), or other formalisms — namely, local combination of goals cannot guarantee global optimality, and it may produce local optima. While a number of devices have been developed, a general solution to these problems must rely on a global analysis (e.g., path planning), with all the computational and informational costs involved.

Our technique shares with the potential field methods the intuition of describing basic units of control as local preferences, expressed on a continuous scale, and then combining these preferences to build complex controls. On the technical level, there are a couple of important differences, both originating in the use of multivalued logic to express goals and controls. First, while the vectors generated by pseudo-forces are summary descriptions of the preferred control, our desirability functions are assignments of utility values to *each* possible control. When we blend control schemas, these assignments, rather than a summary description, are combined into a joint desirability function. In other words, the combined preferred controls are, for us, the preferred controls of the combined preference criterium, and not the combination of the controls individually preferred by each criterium. This

difference in the combination strategy may produce different results when the individual desirability functions are not simple distributions centered around one preferred value. An advantage of our solution is that the combination technique itself is derived from a utility-based semantics for multivalued logics [Ruspini, 1991b]. This allows us to specify in a precise sense the behavior of composed controls, as well as the consistency conditions required for the composition.

The second difference between our technique and potential fields methods, again originating in the use of multivalued logics, 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 used in the combination as a logical proposition gives us more control over the way combination is performed in different situations, and provides a hook to higher-level symbolic processes. We have seen in Section 4 how this ability can help to mitigate the problem of local minima. And Section 5 has shown that a planner can produce complex controls by associating the right context to each behavior.

As an illustration of the similarities and differences between our approach and the potential field methods, consider the AuRA system developed by Arkin [Arkin, 1987; Arkin, 1990a]. Driving his inspiration from studies on biological systems, Arkin adopts a notion of motor schema closely related to our control schemas. One conceptual difference is that Arkin does not distinguish between a motor schema and the behavior that this schema produces when executed by an agent embedded in an environment. Apart from this, control schemas and Arkin's motor schemas have a number of characteristics in common: they both can react directly to sensory information; they both tolerate some uncertainty in perception; they both can be composed into complex behaviors; and they both can be computed in real time and possibly in a distributed fashion. Arkin's schemas, however, are based on a potential field methodology: in every situation, each active motor schema computes a vector representing the most desired speed and direction for that schema; all these vectors are summed and normalized, and the resulting vector is taken to represent some tradeoff control. This way of combining motor schemas has been found to work well in many practical situations. However, Arkin is more interested in the psychological and physiological plausibility of his schemas, and does not engage in a formal analysis of the properties of the composition of behaviors, or on the assumptions of this composition. In fact, as motor schemas are not given semantics in terms of desirability of actions or utility for goals, this formal analysis might be more difficult for Arkin's schemas than it is for our framework.

It is interesting to look at the role that planning plays in our framework and in AuRA. Although both Arkin and ourselves make use of planning techniques, our approaches to the integration of planning and control differ significantly. AuRA's planner is basically a path planner that generates a piece-wise linear path to the goal. Motor schemas to follow this path are dynamically chosen and instantiated at execution time, leg by leg. In our framework, by contrast, the planner generates complex compositions of control schemas, expected to guide the agent toward the achievement of given goals. We do not make strong hypotheses about the planning technology used to generate controls — in fact, we have shown examples where we used two extremely different planning regimes. The fact that behaviors can be described at the high level of abstraction typically used by most current AI planners further supports our claim of generality. The key to this generality is the use of contexts in our basic blending mechanism, and the fact that these contexts are represented in a logical form. In this respect, we notice that behavior combination in AuRA is not weighted by a context of applicability: motor schemas are either active or inactive. Weighted contextual conditions can be encoded in the potential field functions, but this implicit representation does not seem amenable to automatic generation of complex controls.[19]

Of course, there is more to AuRA than motor schemas and their combination. One particularly interesting feature is the way action-oriented perception is modeled in AuRA. Arkin associates a motor schema with perceptual schemas, or *affordances*, that satisfy the perceptual need of that motor schema. Affordances use sensors and interpretation processes to extract the particular type of information needed for a particular schema, and at the level of abstraction that is needed. This solution, again inspired by the research on biological systems, contrasts with classical model-based approaches where one single world model is constructed and maintained. There are, here too, some similarities with our framework. Our control schemas take information from the Local Perceptual Space at whatever level of abstraction is needed by that schema, and different schemas can access the same information but interpret it in different ways. For example, an artifact representing a wall can be treated as an obstacle — having certain geometric properties — by an obstacle avoidance schema; and as an abstract direction to follow — whose geometric details are irrelevant — by a follow-wall schema. However, and unlike Arkin, we still do not have incorporated schemas whose task is to direct the perceptual processes. In this respect, we plan to incorporate some of Arkin's suggestions in our framework.

[19] A mechanism for dynamically modifying the gain of motor schemas based on inter-schema communication was suggested in [Arkin, 1987], but it does not seem to have been incorporated in AuRA.

6.4.2 Circuits methods

An alternative way to define the transfer function of a low-level controller is by means of logical circuits. We have already discussed some examples of systems that define complex controllers in terms of (virtual) circuits: Kaelbling's REX, used in conjunction with GAPPS, and Nilsson's TR-trees. We have also mentioned Connell's architecture, tailored on Brooks's one: his architecture is defined in terms of finite-state machines linked by single bit communications. Connell's architecture differs from the previous two cases because it has been implemented using hardware circuitry [Connell, 1990].

One further system that uses circuits to define transfer functions is the ALFA system proposed by Gat [Gat, 1991]. ALFA is a modular system composed of combinations of two types of circuits, called "modules" and "channels", respectively. Each module is written as a procedure using primitive operators interpreted as circuit components; modules can incorporate a local state, although Gat advocates the use of stateless transfer functions. The outputs of modules are collected together by the channels. These define the strategies to be used to combine outputs. As ALFA allows both digital and analogical components, there are is great variety of possible combination patterns, including averaging and prioritized combination. Gat also built a compiler to translate descriptions given in the ALFA language into (among others) code for a microcontroller.

ALFA is a very general language for building controllers. For example, the ability to define analogical transfer function and averaged combination should allow ALFA to simulate a potential field approach, where each pseudo-force is defined by a module that computes the corresponding vector. However, and like the potential fields, ALFA's modules are functions that map each input configuration to one control value, rather than a preference over controls; hence, ALFA cannot simulate our control schemas. Apart from this, the main difference between Gat's approach and ours is in the emphasis. While Gat seems to be mainly concerned with the generality and expressiveness of his language, and so ends up with a fairly rich and complex system, we were mainly occupied in giving control schemas a simple semantics that enabled us to prove formal properties of the complex controllers we build. (However, as an interesting side effect, control schemas can be implemented by the extremely expressive and general language of fuzzy rules).

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 comport, may lie far away in time and space. By planning, an agent tries to connect the action it performs to its global goals and desires. But the result of planning, the *plan*, has to become physical action. The relation between an abstract representation of a plan and the physical actions that it may induce in the agent has been of interest in recent years, partially influenced by the wider availability of mobile robots. The proposal we have described in this paper is a contribution to this work.

We started from the level of physical movements. Borrowing from control theory, we have defined types of movements as *control schemas*. However, control schemas are less committal than classical controllers, in that they do not map input states to effector commands (or controls), but to a measure of desirability over the space of these commands, the idea being that many commands can generate, to a greater or lesser extent, the same type of movement. To make this precise, we have set up our control schemas in the framework of multivalued logic.

The multivalued nature of our control schemas is the key to our next two steps. On the one hand, we have defined how simpler control schemas can be composed into more complex ones. As control schemas are measures of preference, all we had to do to blend two schemas together was to look for the common preferences: again, multivalued logic provides us with the tools to do this. On the other hand, 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, that is, the anchoring of (the relevant part of) the agent's internal state to external objects through perception, and the notion of *context*, or circumstances, of execution. Our actions, or *behaviors*, are the results of executing control schemas in an environment under certain circumstances. Again, a behavior does not uniquely identify one sequence of movements, or execution, but a set of executions that are, to some measurable extent, possible.

Finally, we have linked behaviors to *goals*, expressed as sets of satisfactory executions. The *good behaviors* for a goal are those that, when executed under their own context, produce executions that satisfy the goal. And, it is especially important that we have shown that under certain hypotheses, by composing behaviors we obtain a new behavior that is good for the composition of the corresponding goals. This result is the end of our story on the definition of complex behavior, and the beginning of the story on the automatic

synthesis of behavior. The composition operators we use are in the orthodoxy of AI approaches to planning — basically, a conjunction operator and a chaining, or sequencing, operator. Moreover, the context of applicability and the goals of our behaviors are defined in a (multivalued) logical language, again in the AI orthodoxy. This suggests that traditional AI techniques for deliberation and means-ends analysis can be readily adapted to generate complex controllers for given (strategic or tactic) goals. And, in fact, we have presented experiments where we used a simple goal-regression planner, and a run-time goal-reduction deliberator to generate complex behaviors, or *embedded plans*, for our testbed mobile robot, Flakey.

The complex control 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. The properties enumerated above point to some significant advantages in using this methodology.

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 [Ruspini and Saffiotti, 1993]. A second important issue that we only touched in this paper is the dynamic modification of complex behaviors, or embedded plans. 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. Finally, although we have shown how we can use artifact and perceptual behaviors to relate perception and action, we feel that much more work is needed in this direction.

The approach that we have described in this paper has been implemented on the SRI mobile robot, Flakey. Most of the illustrative examples presented throughout the paper were taken from this implementation. The performance of Flakey in navigating in an unstructured real-world environment has been demonstrated on various occasions, including innumerable runs in the corridors and offices of SRI's AI Center during working hours, pamphlet distribution during crowded coffee breaks at the second IEEE Conference on Fuzzy Sets and Systems (San Francisco, CA, April 1993), and Flakey's second place at the first robotic competition of the AAAI (San Jose, CA, July 1992) [Congdon *et al.*, 1993].

Acknowledgments Nicolas Helft, David Israel and Daniela Musto contributed to the development of the ideas presented in this paper.

References

[Akar and Umit, 1990] L. Akar and O. Umit. Design of knowledge-rich hierarchical controllers for large functional systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(4):791–803, 1990.

[Arkin, 1987] 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.

[Arkin, 1990a] R. C. Arkin. The impact of cybernetics on the design of a mobile robot system: a case study. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(6):1245–1257, 1990.

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

[Bajcsy, 1988] R. Bajcsy. Active perception. In *Proceedings of the IEEE*, volume 76, pages 966–1005, 1988.

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

[Benson, 1993] S. Benson. Unpublished manuscript. Technical report, Stanford University, Robotics Laboratory, Stanford, CA, 1993.

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

[Brooks, 1987] R. Brooks. Intelligence without representation. In *Procs. of the Workshop on the Foudations of AI*, MIT, Cambridge, MA, 1987.

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

[Congdon *et al.*, 1993] 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.

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

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

[Dean and Wellman, 1991] T. L. Dean and M. P. Wellman. *Planning and control.* Morgan Kaufmann, San Mateo, CA, 1991.

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

[Fikes and Nilsson, 1971] 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.

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

[Firby, 1989] J. R. Firby. Adaptive execution in complex dynamic worlds. Technical Report 672, Dept. of Computer Science, Yale University, 1989.

[Gat, 1991] E. Gat. *Reliable Goal-Directed Reactive Control for Real-World Autonomous Mobile Robots.* PhD thesis, Virginia Polytechnic Institute and State University, 1991.

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

[Georgeff and Ingrand, 1989] 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.

[Georgeff, 1987] M. Georgeff. Planning. Technical Report 418, SRI Artificial Intelligence Center, 1987.

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

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

[Kaelbling, 1987] L. P. Kaelbling. An architecture for intelligent reactive systems. In M.P. Georgeff and A.L. Lansky, editors, *Reasoning about Actions and Plans.* Morgan Kaufmann, 1987.

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

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

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

[Klir and Folger, 1988] G. J. Klir and T. A. Folger. *Fuzzy sets, uncertainty, and information.* Prentice-Hall, 1988.

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

[Lee, 1990] C. C. Lee. Fuzzy locin in control systems: fuzzy logic controller (Parts I and II). *IEEE Trans. on Systems, Man, and Cybernetics*, 20(2):404–435, 1990.

[Łukasiewicz and Tarski, 1983] 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.

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

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

[McDermott, 1990b] Drew McDermott. Planning reactive behavior: a progress report. In *Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 450–458, 1990.

[Nilsson, 1980] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, California, 1980.

[Nilsson, 1984] Nils J. Nilsson. SHAKEY the robot. Technical Note 323, SRI Artificial Intelligence Center, Menlo Park, California, 1984.

[Nilsson, 1985] N. J. Nilsson. Triangle tables: A proposal for a robot programming language. Technical Report 347, SRI International, 1985.

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

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

[Payton, 1990] D. W. Payton. Exploiting plans as resources for action. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.

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

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

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

[Ruspini and Saffiotti, 1993] E. H. Ruspini and A. Saffiotti. The formulation and solution of non-linear optimization problems in multi-valued logic. Technical report, SRI Artificial Intelligence Center, Menlo Park, California, 1993. Forthcoming.

[Ruspini, 1990] 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.

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

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

[Saffiotti *et al.*, 1993a] 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*, San Francisco, CA, 1993.

[Saffiotti *et al.*, 1993b] A. Saffiotti, E. H. Ruspini, and K. Konolige. Robust control of a mobile robot using fuzzy logic. In *Procs. of the European Congress on Fuzzy and Intelligent Technologies*, Aachen, Germany, 1993.

[Saffiotti *et al.*, 1993c] A. Saffiotti, E. H. RuspiniK., and Konolige. A fuzzy controller for Flakey, an autonomous mobile robot. Technical Report 529, SRI Artificial Intelligence Center, Menlo Park, California, 1993.

[Saffiotti, 1993] A. Saffiotti. Some notes on the integration of planning and reactivity in autonomous mobile robots. In *Procs. of the AAAI Spring Symposium on Foundations of Automatic Planning*, pages 122–126, Stanford, CA, 1993.

[Schoppers, 1987] 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.

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

[Simmons, 1990] R. Simmons. An architecture for coordinating planning, sensing and action. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.

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

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

[Yen and Pfluger, 1992] 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.

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

Appendix: Proofs

THEOREM 2.1

Let D_1 and D_2 be two desirability functions, and t any trajectory in S^ω. Then

$$Traj_{(D_1 \otimes D_2)}(t) \leq Traj_{D_1}(t) \circledast Traj_{D_2}(t)$$

Proof. For any trajectory $t = (s_0, s_1, \ldots, s_k)$, $k > 0$:

$$\begin{aligned}
&Traj_{(D_1 \otimes D_2)}(t) = \\
&= \inf_{0 \leq i < k} Next_{(D_1 \otimes D_2)}(s_i, s_{i+1}) \\
&= \inf_{0 \leq i < k} \sup_{a \in A} [(D_1(s_i, a) \circledast D_2(s_i, a)) \circledast M(s_i, a, s_{i+1})] \\
&\quad \text{by exploiting the fact that our t-norm is idempotent:} \\
&= \inf_{0 \leq i < k} \sup_{a \in A} [(D_1(s_i, a) \circledast D_2(s_i, a)) \circledast M(s_i, a, s_{i+1})] \circledast \\
&\quad \inf_{0 \leq i < k} \sup_{a \in A} [(D_1(s_i, a) \circledast D_2(s_i, a)) \circledast M(s_i, a, s_{i+1})] \\
&\leq \inf_{0 \leq i < k} \sup_{a \in A} [D_1(s_i, a) \circledast M(s_i, a, s_{i+1})] \circledast \\
&\quad \inf_{0 \leq i < k} \sup_{a \in A} [D_2(s_i, a) \circledast M(s_i, a, s_{i+1})] \\
&= [\inf_{0 \leq i < k} Next_{D_1}(s_i, s_{i+1})] \circledast [\inf_{0 \leq i < k} Next_{D_2}(s_i, s_{i+1})] \\
&= Traj_{D_1}(t) \circledast Traj_{D_2}(t)
\end{aligned}$$

□

THEOREM 2.2

Let D be a desirability function, and C a context on S. Then

$$Traj_{D|C}(t) \leq \sup_{t' \subseteq t} [\, Traj_D(t') \oslash C(t') \,]$$

Proof. Let $t = (s_0, s_1, \ldots, s_k)$, $k > 0$, and consider any subtrajectory t' of t: that is, $t' = (s_j, s_{j+1}, \ldots, s_h)$, $0 \leq j < h \leq k$. Then:

$$\begin{aligned}
Traj_D(t') \oslash C(t') &= \inf_{j \leq i < h} Next_D(s_i, s_{i+1}) \oslash C(t') \\
&= \inf_{j \leq i < h} \sup_{a \in A} (D(s_i, a) \circledast M(s_i, a, s_{i+1})) \oslash \inf_{j \leq i < h} C(s_i)
\end{aligned}$$

$$
\begin{aligned}
&\geq \inf_{0\leq i<k} \sup_{a\in A} (D(s_i,a) \circledast M(s_i,a,s_{i+1})) \oslash \inf_{0\leq i<k} C(s_i) \\
&\quad \text{as } C(s_i) \text{ does not depend on } a \\
&\geq \inf_{0\leq i<k} \sup_{a\in A} [(D(s_i,a) \circledast M(s_i,a,s_{i+1})) \oslash C(s_i)] \\
&\geq \inf_{0\leq i<k} \sup_{a\in A} [(D(s_i,a) \oslash C(s_i)) \circledast (M(s_i,a,s_{i+1}) \oslash C(s_i))] \\
&\geq \inf_{0\leq i<k} \sup_{a\in A} [(D(s_i,a) \oslash C(s_i)) \circledast M(s_i,a,s_{i+1})] \\
&= \inf_{0\leq i<k} \sup_{a\in A} [(D^{\downarrow C}(s_i,a) \circledast M(s_i,a,s_{i+1})] \\
&= \inf_{0\leq i<k} Next_{D\downarrow C}(s_i,s_{i+1}) \\
&= Traj_{D\downarrow C}(t)
\end{aligned}
$$

As this holds for any t', we have the thesis. □

THEOREM 4.1 (CONJUNCTION OF BEHAVIORS)
Let $B_1 = (C_1, D_1, O_1\rangle$ and $B_2 = (C_2, D_2, O_2\rangle$ be two control structures, and let G_1, G_2 be two goals. Then, if $Good(B_1, G_1) \geq \alpha$ and $Good(B_2, G_2) \geq \beta$, then

$$Good(\text{CONJ}(B_1, B_2), G_1 \otimes G_2) \geq \alpha \circledast \beta .$$

Proof. Let $\Psi(t)$ denote $[Traj_{(D_1 \otimes D_2)}(t) \circledast C_1(t) \circledast C_2(t)]$. That is, $\Psi(t)$ identifies the trajectories that need to be considered by $Good[(\text{CONJ}(B_1, B_2)), (G_1 \otimes G_2)]$. Note that, because of Theorem 3.3, for any t, $\Psi(t) \leq (Traj_{D_1}(t) \circledast C_1(t))$ and $\Psi(t) \leq (Traj_{D_2}(t) \circledast C_2(t))$. Then:

$$
\begin{aligned}
&Good(\text{CONJ}(B_1, B_2), G_1 \otimes G_2) = \\
&= \inf_{t\in S^\omega} [(G_1(t) \circledast G_2(t)) \oslash \Psi(t)] \\
&\geq \inf_{t\in S^\omega} [(G_1(t) \oslash \Psi(t)) \circledast (G_2(t) \oslash \Psi(t))] \\
&\geq \inf_{t\in S^\omega} (G_1(t) \oslash \Psi(t)) \circledast \inf_{t\in S^\omega} (G_2(t) \oslash \Psi(t)) \\
&\geq \inf_{t\in S^\omega} [G_1(t) \oslash (Traj_{D_1}(t) \circledast C_1(t))] \circledast \inf_{t\in S^\omega} [G_2(t) \oslash (Traj_{D_2}(t) \circledast C_2(t))] \\
&= Good(B_1, G_1) \circledast Good(B_2, G_2)
\end{aligned}
$$

□

PROPOSITION 4.2 *Let G_1, G_2 be two goals, abd B be any control structure. Then*

$$Good(B, G_1) \circledast Good(B, G_2) \leq Cons_G(G_1, G_2)\,.$$

Proof. The proof is straightforward, and was already outlined in the text. □

THEOREM 4.3 (DISJUNCTION OF BEHAVIORS)
Let $B_1 = \langle C_1, D_1, O_1 \rangle$ and $B_2 = \langle C_2, D_2, O_2 \rangle$ be two control structures, and let G_1, G_2 be two goals. Then, if $Good(B_1, G_1) \geq \alpha$ and $Good(B_2, G_2) \geq \beta$, then

$$Good[\text{DISJ}(B_1, B_2),\ G_1 \oplus G_2] \geq \alpha \circledast \beta\,.$$

where, for all t, $(G_1 \oplus G_2)(t) = G_1(t) \oplus G_2(t)$.

Proof. Let

$$\Psi_1(t) = Traj_{(D_1^{\downarrow C_1} \otimes D_2^{\downarrow C_2})}(t) \circledast C_1(t)$$

and

$$\Psi_2(t) = Traj_{(D_1^{\downarrow C_1} \otimes D_2^{\downarrow C_2})}(t) \circledast C_2(t)\,.$$

Note that $\Psi_1(t) \leq Traj_{D_1^{\downarrow C_1}}(t) \circledast C_1(t)$ and similarly for Ψ_2. Then:

$Good(\text{DISJ}(B_1, B_2),\ G_1 \oplus G_2) =$

$$\begin{aligned}
&= \inf_{t \in S^\omega} [(G_1(t) \oplus G_2(t)) \oslash (Traj_{(D_1^{\downarrow C_1} \otimes D_2^{\downarrow C_2})}(t) \circledast (C_1(t) \oplus C_2(t)))] \\
&= \inf_{t \in S^\omega} [(G_1(t) \oplus G_2(t)) \oslash (\Psi_1(t) \oplus \Psi_2(t))] \\
&= \inf_{t \in S^\omega} [((G_1(t) \oplus G_2(t)) \oslash \Psi_1(t)) \circledast ((G_1(t) \circledast G_2(t)) \oslash \Psi_2(t))] \\
&\geq \inf_{t \in S^\omega} [(G_1(t) \oslash \Psi_1(t)) \circledast (G_2(t) \oslash \Psi_2(t))] \\
&\geq \inf_{t \in S^\omega} (G_1(t) \oslash \Psi_1(t)) \circledast \inf_{t \in S^\omega} (G_2(t) \oslash \Psi_2(t)) \\
&\geq \inf_{t \in S^\omega} [G_1(t) \oslash (Traj_{D_1^{\downarrow C_1}}(t) \circledast C_1(t))] \circledast \inf_{t \in S^\omega} [G_2(t) \oslash (Traj_{D_2^{\downarrow C_2}}(t) \circledast C_2(t))] \\
&= Good(B_1, G_1) \circledast Good(B_2, G_2)
\end{aligned}$$

□

THEOREM 4.4 (CHAINING OF BEHAVIORS)
Let $B_1 = \langle C_1, D_1, O_1 \rangle$ and $B_2 = \langle C_2, D_2, O_2 \rangle$ be two control structures, and let G be a goal. Then, if $Good(B_1, \text{ACHIEVE}[C_2]) \geq \alpha$ and $Eventually(B_2, G) \geq \beta$, then

$$Eventually(\text{CHAIN}(B_1, B_2),\ G) \geq \alpha \circledast \beta\,.$$

To simplify the proof of this theorem, we first demonstrate the following:

LEMMA 4.5 *Under the hypotheses of Theorem 4.4, for every t there is a $t' \in \text{tail}(t)$ such that*

$$Traj_{\text{CHAIN}(B_1,B_2)}(t) \circledast (C_1 \cup C_2)(t) \leq (C_2(t') \circledast Traj_{B_2}(t')) \oplus (C_2(t') \oslash \alpha)$$

Proof. We start by considering the effect of $Good(B_1, \text{ACHIEVE}[C_2]) \geq \alpha$ over the set of trajectories of $B_1^{\downarrow C_1 \setminus C_2}$. We have:

$$\begin{aligned} & Good(B_1, \text{ACHIEVE}[C_2]) \\ & \quad = \inf_{t \in S^\omega} [\text{ACHIEVE}[C_2](t) \oslash (Traj_{D_1}(t) \circledast C_1(t))] \\ & \quad \leq \inf_{t \in S^\omega} [\text{ACHIEVE}[C_2](t) \oslash (Traj_{D_1^{\downarrow C_1 \setminus C_2}}(t) \circledast (C_1 \setminus C_2)(t))] \end{aligned}$$

Then, for any trajectory $t = (s_0, s_1, \ldots, s_k)$, $k > 0$, we have:

$$\begin{aligned} Traj_{D_1^{\downarrow C_1 \setminus C_2}}(t) \circledast (C_1 \setminus C_2)(t) & \leq \text{ACHIEVE}[C_2](t) \oslash Good(B_1, \text{ACHIEVE}[C_2]) \\ & \leq \text{ACHIEVE}[C_2](t) \oslash \alpha \\ & = \sup_{s \in t} C_2(t) \oslash \alpha \end{aligned}$$

We now use this constraint to characterize the admissible trajectories for the $\text{CHAIN}(B_1, B_2)$ given the hypotheses. For any t:

$$\begin{aligned} & Traj_{\text{CHAIN}(B_1,B_2)}(t) \\ & \quad = \inf_{0 \leq i < k} Next_{(D_1^{\downarrow C_1 \setminus C_2} \otimes D_2^{\downarrow C_2})}(s_i, s_{i+1}) \\ & \quad = \inf_{0 \leq i < k} \sup_{a \in A} [(D_1^{\downarrow C_1 \setminus C_2}(s_i, a) \circledast D_2^{\downarrow C_2}(s_i, a)) \circledast M(s_i, a, s_{i+1})] \end{aligned}$$

We denote by $\Psi(t)$ the last espression. We have, for any t:

$$\Psi(t) \leq \inf_{0 \leq i < k} \sup_{a \in A} (D_1^{\downarrow C_1 \setminus C_2}(s_i, a) \circledast M(s_i, a, s_{i+1}))$$

and analogously

$$\Psi(t) \leq \inf_{0 \leq i < k} \sup_{a \in A} (D_2^{\downarrow C_2}(s_i, a) \circledast M(s_i, a, s_{i+1})) = Traj_{D_2^{\downarrow C_2}}(t)$$

Consider now

$$
\begin{aligned}
& Traj_{\text{CHAIN}(B_1,B_2)}(t) \circledast (C_1 \cup C_2)(t) \\
&= \Psi(t) \circledast (C_1 \cup C_2)(t) \\
&= (\Psi(t) \circledast (C_1 \setminus C_2)(t)) \oplus (\Psi(t) \circledast C_2(t)) \\
&\leq [\inf_{0 \leq i < k} \sup_{a \in A} (D_1^{\downarrow C_1 \setminus C_2}(s_i, a) \circledast M(s_i, a, s_{i+1})) \circledast (C_1 \setminus C_2)(t)] \\
&\quad \oplus [Traj_{D_2^{\downarrow C_2}}(t) \circledast C_2(t)] \\
&\leq [\sup_{s \in t} C_2(t) \oslash \alpha] \oplus [Traj_{D_2^{\downarrow C_2}}(t) \circledast C_2(t)]
\end{aligned}
$$

Consider now first index i in $t = (s_0, s_1, \ldots, s_k)$ for which $C_2(S_i) \geq \alpha$. As $Good(B_1, \text{ACHIEVE}[C_2]) \geq \alpha$, there must be such an index. Call this index $\hat{\imath}$, and consider the subsequence t' of t given by: $t' = (s_{\hat{\imath}}, s_{\hat{\imath}+1}, \ldots, s_k)$. We have:

$$
(\sup_{s \in t'} C_2(t') \oslash \alpha) \oplus (Traj_{D_2^{\downarrow C_2}}(t) \circledast C_2(t)) \leq (C_2(t') \oslash \alpha) \oplus (Traj_{D_2^{\downarrow C_2}}(t) \circledast C_2(t))
$$

Hence the thesis. □

Proof. [of Theorem 4.4] The hypothesis $Eventually(B_2, G) \geq \beta$, means that

$$
\beta \leq \inf_{t \in S^\omega} \sup_{t' \in \text{tail}(t)} [G(t) \oslash (Traj_{B_2}(t) \circledast C_2(t))]. \tag{30}
$$

This means that for every trajectory $t = (s_0, s_1, \ldots, s_k), k > 0$ we can find a $0 < j < k$ such that for any subtrajectory $t'' = (s_j, s_{j+1}, \ldots, s_k)$ it is:

$$
G(t'') \oslash (Traj_{B_2}(t'') \circledast C_2(t'')) \geq \beta .
$$

Take now any trajectory t, and consider the subtrajectory $t' \in \text{tail}(t)$ used in Lemma 4.5: as $t' \in \text{tail}(t)$, there is a $0 < h < k$ such that $t' = (s_h, s_{h+1}, \ldots, s_k)$. Let $m = \max(j, h)$, and consider $\hat{t} = (s_m, s_{m+1}, \ldots, s_k)$. $\hat{t}$ has both the property (30) and the property stated in Lemma 4.5. Consider now:

$$
\begin{aligned}
& Good(\text{CHAIN}(B_1, B_2), G) = \\
&\quad = \inf_{t \in S^\omega} \sup_{t' \in \text{tail}(t)} [G(t) \oslash (Traj_{\text{CHAIN}(B_1,B_2)}(t) \circledast (C_1 \cup C_2)(t))]
\end{aligned}
$$

By using $\hat{t}$, we have:

$$
[G(t) \oslash (Traj_{\text{CHAIN}(B_1,B_2)}(t) \circledast (C_1 \cup C_2)(t))] \geq
$$

$$
\begin{aligned}
&\geq G(\hat{t}) \oslash [(C_2(\hat{t}) \circledast Traj_{B_2}(\hat{t})) \oplus (C_2(\hat{t}) \oslash \alpha))] \\
&= [G(\hat{t}) \oslash (C_2(\hat{t}) \circledast Traj_{B_2}(\hat{t}))] \circledast [G(\hat{t}) \oslash (C_2(\hat{t}) \oslash \alpha)] \\
&\geq \beta \circledast [G(\hat{t}) \oslash (C_2(\hat{t}) \oslash \alpha)] \\
&= \beta \circledast [(G(\hat{t}) \circledast \alpha) \oslash C_2(\hat{t})] \\
&\geq \beta \circledast (\beta \circledast \alpha)
\end{aligned}
$$

As we can find such a $\hat{t}$ for any trajectory t, we have:

$$
Good(\text{CHAIN}(B_1, B_2), G) \geq \beta \circledast \beta \circledast \alpha \tag{31}
$$

Equation (31) is the general form of our thesis. When $\circledast$ is idempotent (which is the case of the min operator used in this paper), we have the form of the thesis used in our statement. □