

Towards Synthesis of Reactive & Robust Behavior Chains

Amol D. Mali

Electrical Engg. & Computer Science, P.O.Box 784
University of Wisconsin, Milwaukee,
Milwaukee, WI 53201, USA.
e-mail: mali@miller.cs.uwm.edu
Phone: 1-414-906-8942, Fax: 1-414-229-2769

Abstract

(Appears in the proceedings of the international conference on artificial intelligence and soft computing (ASC), Canada, July 2000, pp. 508-515.)

It has been argued that to successfully function in highly dynamic and uncertain environments, autonomous robots need to be reactive and robust. Behavior-based robots that identify and repair the failures have been considered to be a promising option to achieve such a functionality. However, the notions of reactivity and robustness have been hitherto only informally used and have been loaded with various interpretations. This has prevented the design and synthesis of chains of behaviors whose reactivity and robustness can be controlled. We define reactivity and robustness and show how methods for synthesizing behavior chains with controllable reactivity and robustness can be developed. These design and synthesis techniques lead to a new computational architecture for the autonomous robots that need to be both reactive and robust.

Keywords - Autonomy, Robotics, Behaviors, Reactivity, Robustness.

1 Introduction

Classical planners make a number of simplifying assumptions like static and completely observable environment, deterministic actions and perfect perception. Challenging this operation in closed world, behavior-based robots that autonomously interact with their environments and avoid global representations were proposed and developed [6]. However, these reactive robots exhibited emergent behavior that was sometimes undesirable. Since these robots were situation-driven and did no goal directed reasoning like backward chaining planners, these robots were viewed as a combination of lack of goal directedness and incompleteness of depth first forward search. Hence these robots were augmented with planning capabilities. Since autonomous robots may fail in a variety of ways in dynamic and uncertain environments, a variety of mod-

ules for monitoring an environment, managing the contention for sensors, monitoring the internal conditions of a robot and even learning modules to improve functionality using the domain models, failures and successes were added. Though the introduction of these modules has its own merits and motivation, the transition towards the implementation of these complex architectures has left many important ideas in an informal stage. Rosenschein & Kaelbling [23] have acknowledged that though reactive architectures have been proposed as an alternative to traditional AI, their theoretical foundations are less developed. In particular, the widely used key ideas like *reactivity* and *robustness* never received a precise treatment. We visit the conventional interpretations of reactivity and robustness and show that though useful, they do not lead to design and synthesis of complex behavior whose reactivity and robustness can be controlled. To rectify this situation, we define metrics for reactivity and robustness. Then we show how these metrics can be used to synthesize behavior chains whose reactivity and robustness can be controlled. We propose a new architecture for autonomous robots that uses such chains. This architecture consists of a set of behaviors, a set of behavior chains and a chain modification mechanism for fulfilling the tasks.

This paper is organized as follows. We give an overview of the current architectures of autonomous robots in section 2 and discuss the existing interpretations of robustness and reactivity in section 3. In section 4, we build a model of reactive and robust functionality. In section 5, we show how one can synthesize behavior chains whose robustness and reactivity can be controlled, using the design foundations in section 4. In section 6, we suggest a number of methods to significantly reduce the complexity of the synthesis and modification of these chains to make this approach practical. We also discuss some relevant insights from the classical planning literature in this section. Our conclusions are presented in section 7.

2 Background

We review the successes and limitations of the three types of autonomous robot architectures in this section - reactive, deliberative and hybrid, in this order. This classification covers most of the existing architectures, except those that contain a learning component. Cyclic wandering, a typical characteristic of purely reactive behaviors, is reported in [2]. Behavior-based box pushing robots [19] can get stuck in cycles. Connell [9] reports cyclic behavior of a can collection robot. While commenting on such reactive systems, Kirsh [18] cogently argues that the duplication of insect behaviors like wandering, avoiding obstacles and following corridors does not prove that behavior-based robotics is a “royal path to high level behaviors”.

The world of Shakey [10] was very carefully engineered. Walls were specially constructed and painted with matte finishes. This deliberative robot computed and executed plans for goals, but its knowledge of the world was stored in its knowledge base. As Agre [1] says, just as planning offered no account of moment-to-moment interaction with the world, reaction offered no account of how the organism or robot’s actions could be guaranteed to work. This concern inspired the development of the hybrid architectures.

Saffiotti et al [24] and Arkin [3] integrate planning and reactive control, for robot navigation. Many more implemented hybrid architectures like [11],[13],[25] exist. A rover Rockey-III that navigates through rough outdoor terrain and collects soil samples has been reported in [21]. Bonasso et al [5] describe a robot for find and fetch tasks in an outdoor environment.

Despite these interesting implementations, a number of problems exist. Planner and reactor are used in ad-hoc ways. Planner is used all the time and reactions are used for exceptional situations [25]. This is contrary to the widespread strategy of using reactor as much as possible and planner only to handle problems with reactivity. Chapman [7] says that combining classical planners with reactive systems combines the worst of both worlds - planning is expensive and reactivity is myopic. It is concluded by Firby & Simmons [12] that hybrid architectures do not combine reactive and deliberative paradigms effectively. Though a lot of implementations have acknowledged and addressed the need to have robust and reactive functionality, the precise treatment that can make the synthesis of behavior chains with controllable reactivity and robustness possible is missing.

3 Robustness & Reactivity

We here examine the conventional interpretations of the notions of robustness and reactivity.

3.1 Robustness

There are at least five different interpretations of the notion of robustness that we visit here.

(a) Graceful Degradation The behaviors of a robot can be organized into a hierarchy depending on their complexity, as in the subsumption architecture of Brooks [6]. For example, solving a puzzle may be viewed as more complex than avoiding an obstacle, as per some complexity metric like the amount of search required to solve the problem. Each behavior in the hierarchy needs some resources (computational or physical) to function successfully. For example, efficiently solving the puzzle will require good heuristics and avoiding obstacles will require sonars whose readings are not arbitrarily noisy. As the availability of the resources or the quality of the pre-requisites deteriorates, the corresponding behaviors degrade. However, the robot continues to have a mobile existence in the world until the pre-requisite of the simplest behavior is lost, degrading gracefully.

(b) Repeated Effort This exhibits a strong commitment to achieving the goals. The autonomous robots that react to external stimuli do partially have this feature, since they continue to respond to a stimulus as long as it is true. If the can falls down at an inappropriate location because of a slippery gripper, the robot will pick it up again, as long as the can is visible. However if the can rolls on the floor and reaches an area far away, the robot will have to track the can and move it. Such a “try until you succeed” type behavior is also called robust.

(c) Pre-canned Methods This can be considered as achieving fault tolerance through some sort of redundancy in the software. Important or frequently occurring failures and the situations under which they occur are identified *a priori* and a set of situation-dependent remedies are supplied to the robot for online use. Even alternative methods (hence the redundancy in the software) to fix the same failure in the same situation may be supplied, e.g. avoiding obstacles in a dark room can be handled by both sophisticated image processing and a sonar-based method. Thus the complexity of online synthesis of a remedy is pushed to the complexity of offline design. Because of the ability to handle multiple failures under multiple situations, the behavior looks robust.

(d) Redundant Hardware Extra sensors can be provided to avoid the contention for sensory resources arising in perceptually demanding situations, e.g. navigation in the presence of multiple mobile obstacles. In this case the robot may want to use the camera to pick up an object but an obstacle approaching may force the camera to be turned in a different direction. Similarly, if sonars are used to detect obstacles of certain height,

obstacles that are too short may go undetected. Such situations can be handled by providing more cameras and installing additional sonars at the base of the robot to handle small obstacles on ground.

(e) General Behavior A behavior can be viewed as robust if it is general enough to work under different circumstances. This can be achieved by providing a weaker or more general stimulus. For example, the behavior of picking with stimulus $\exists x(can(x) \vee bottle(x) \vee cup(x))$ is more general than the behavior with the stimulus $\exists x(can(x) \vee bottle(x))$ because the former behavior can pick up cups that the latter cannot. \exists and \vee denote existential quantifier and logical disjunction respectively.

All these interpretations of robustness, though valuable, focus on equipping a robot to handle failures due to a problem in the hardware or the changes in the external environment. Our view of robustness is more general and it is based on the ability of a behavior space to efficiently deal with changes. These changes could be changes in goals to be achieved or changes in states of hardware, changes in an external environment or changes in the internal conditions of a robot.

3.2 Reactivity

We discuss various interpretations of the term “reactivity” here that are based on the factors like internal state, response time and computational structure.

(a) Amount of Internal State Reactive robots react to the percepts from their environment (e.g. obstacles, gestures of their user etc.) As per the interpretation of reactivity based on this internal state criterion, since the stimuli that trigger the behaviors of a robot are available in the environment itself, higher reactivity means lower internal state and vice versa. However there is no established threshold for the amount of internal state that draws a line between reactive and deliberative behavior. Some argue that reactive behavior maintains no internal state and thus random navigation with obstacle avoidance using bumpers or sonars is perhaps the most complex reactive behavior. On the other hand, Brooks [6] allows limited internal state that is distributed and thus local to the behaviors and he claims that 97 % of the pragmatic activity is reactive [18].

(b) Response Time As per this interpretation, behaviors that respond faster to the stimuli or user’s requests are more reactive. This notion is very close to that of reflex action. For example, a robot that avoids obstacles by using the sonar readings to infer the presence of obstacles may avoid them faster than a robot that grabs the image of its surrounding and then processes it to extract shapes and matches them with the internal models of obstacles. Thus the robot

that uses sonars may be considered to be more reactive than the robot that uses vision. However there are, like the internal state-based interpretation, no established thresholds for response times to create the reactive-deliberative dichotomy. A vision-based robot that uses parallel processors and efficient hardware may respond faster than the sonar-based robot. Hence the response time is an ad-hoc measure.

(c) Computational Structure The production rules with the “if (condition) then (action) else ..” structure have a reactive flavor where the action executed can be viewed as a reaction to the condition that is satisfied. The behavior modules in [6][9] are defined in this manner.

Our explanation of these three ways of understanding the concept of reactivity shows that all of them have been used without formalization. Response time can be viewed as being dependent on the reasoning performed over the internal state. The response time may also be affected by the delays in the hardware, however we focus on the internal computational structure here and ignore the hardware delays, though these considerations are important in practice. We use the internal state and its persistence to define a metric to capture the amount of reactivity. We adopt the traditional production style (*if (condition) then (action) else ..*) computational structure to express the behaviors.

4 A Model of Reactivity & Robustness

In this section, we develop a model of reactive and robust functionality. The notions of reactivity and robustness here are the key design foundations for the synthesis of reactive and robust behavior chains we address in sections 5 and 6.

- **Behavior** - A behavior β_i is modeled as a 2 tuple $\langle s_i, c_i \rangle$ and defined as a mapping from stimulus s_i to consequence c_i . There is an action part between stimulus and consequence which makes the consequence true. Besides motor control, the action part may contain a computational mechanism to update internal states of a behavior.

- **Behavior space** - It is the set of all behaviors of a robot. It is denoted by B and $|B|$ is size of the behavior space.

- **Stimulus** - We assume that stimuli of all behaviors are expressed in a purely conjunctive form. It is assumed that each literal in a stimulus may correspond to a number of sensor readings, i.e. sensor readings are processed to extract meaning out of them and there may be a literal to which this meaning is mapped. (A predicate is an atomic formula and a literal is an atomic formula or negation of an atomic formula. We assume that stimuli contain only positive literals (thus

no negations are allowed in stimuli.)

The universe is a conjunction of literals denoted by U . Hence when we say that a stimulus is s_i , we mean that it is $(s_i \wedge X)$, where $(U \Rightarrow X)$, X being a part of the universe, e.g. to pick up a can, it is necessary for a robot to have a gripper in a good condition (not mechanically damaged), but this is not listed in stimulus of the behavior *pick_up_can*. \Rightarrow and \wedge respectively denote logical implication and logical conjunction respectively.

- **Consequence** - It is assumed that consequences of all behaviors are expressed in a purely conjunctive form. The consequences may contain both positive and negative literals.

- **Behavior chain** - Complex behavior occurs because a number of primitive behaviors (like β_i) operate sequentially and/or concurrently. Here we focus on the temporal sequencing mechanism that gives rise to a complex behavior. A behavior chain C is a temporal sequence of behaviors, $\{\beta_{i_1} : \beta_{i_2} : \beta_{i_3} : \dots : \beta_{i_k}\}$, where $\beta_{i_m} : \beta_{i_{m+1}}$ is used to denote that these two behaviors are contiguous and occur immediately next to each other in time, with the former behavior preceding the latter. Such a chain is said to be composable from the behavior space B if behaviors in the chain are elements of B . This is denoted by $C \triangleleft B$. This sequential model of complex behavior is very widely used in ethological analysis [8]. $\beta_i \prec \beta_j$ denotes the temporal precedence relation where β_i occurs in the chain earlier than β_j . Concurrent behaviors that do not interfere can be ordered (linearized) and included in the sequential chain model, e.g. if the behaviors β_7 and β_{10} occur concurrently, they can either be treated as $\{\beta_7 : \beta_{10}\}$ or $\{\beta_{10} : \beta_7\}$. Our behavior chains are similar to the paths in the finite state diagram of behavior schemas used in the hybrid planner-reactor architecture of Arkin & Balch [4].

In the chaining mechanism, the action of an earlier behavior changes the situation in such a way that the newly changed part of the situation in conjunction with the the universe U implies stimulus for next behavior in the chain (however the universe is not explicitly listed in the logical formulae which describe stimuli and consequences). Consider the chain $\{\beta_1 : \beta_3 : \beta_7\}$. Here we may have $c_1 = (a \wedge c)$ and $s_3 = (a \wedge c \wedge z)$. ($c_1 \Rightarrow s_3$) if $(U \Rightarrow z)$, but z is not explicitly listed in c_1 . In case z is not true in the universe, the chain will be broken.

The length of a chain C_i is defined as the number of behaviors in the chain. Behavior chains are responsible for fulfilling tasks (defined in the discussion on *task space* in this section).

- **Internal State** - The local internal state of a behavior can be considered to be of two types - perma-

nent and temporary. We assume that the permanent component of the internal state is fixed and the temporary component is variable. The internal state of a behavior β_i , denoted by δ_i is thus defined as the function $f_i(\delta_{ip}, \delta_{it})$, where δ_{ip}, δ_{it} denote the permanent and temporary components. The estimates of δ_{ip}, δ_{it} can be provided using the knowledge of the functionality of β_i . The temporary part is variable since it depends upon the context. For example, the behavior of picking dishes from a particular table may store the location of the table and the model of a dish (so that it can be compared with the image grabbed) in the permanent component and the location of the dish to be picked next in the temporary component. This temporary state may be updated as the robot moves (since the distance of the dish from the robot changes), until the dish is picked up. The temporary state may also be updated because the object providing the stimulus is mobile. Not all internal state of a behavior is necessarily a part of its stimulus, e.g. the stimulus of the behavior of picking dishes may be $\exists x(dish(x) \wedge graspable(x))$ and clearly, the distance of the dish from the robot is such a portion of internal state (which does not appear in the stimulus). The idea behind maintaining the temporary internal state and updating it is to avoid sensing from scratch, e.g. if the robot recognizes a cupboard to be opened and if the camera turns in another direction (leaving the cupboard out of sight), it is not a good idea to be recognize the cupboard again, since this will duplicate the recognition computation. Thus the location of the cupboard may be stored in the temporary internal state and updated as the robot moves. It is not just the location of an object providing the stimulus that may be a part of the temporary internal state, some local cues useful in relocating the stimulus may be stored in the temporary internal state as well. We assume that the temporary internal state is erased immediately after the completion of the execution of a behavior.

We consider a stimulus to be true when all predicates from the stimulus are true. The temporary internal state is stored from that point in time when some part of a stimulus is detected to be true, e.g. a from $(a \wedge b \wedge c)$. The difference between this time and the time at which the behavior execution ends is the time for which the temporary internal state is stored, different parts of the temporary internal state detected at different times being stored for different times. Such times can be arbitrarily long and need to be controlled to achieve faster functionality and avoid state update computations. For example, if the stimulus of a behavior is detected to be true at time t and the behavior execution starts at time t' , then it is important to min-

imize the idle time ($t' - t$) by executing the behavior as early as possible, thus controlling the value of t' . The other option is to control the value of t by not recording the state even if the stimulus is available (but this may duplicate the recognition computation). If this is not done, the behavior remains idle.

- **Effective Internal State** - The amount of time for which a state is stored is important besides the state, because of its potential effect on the response time due to the state update computations. We consider these two factors in the notion of the effective internal state of a chain. The effective internal state of a chain $\{\beta_1 : \beta_2 : \beta_3 : \dots : \beta_n\}$ is defined as a function of the internal states of the individual behaviors and the times for which they are stored, $g(f_1(\delta_{1p}, \delta_{1t}), \dots, f_n(\delta_{np}, \delta_{nt}), t_{11}, t_{12}, \dots, t_{21}, t_{22}, \dots, t_{31}, t_{32}, t_{33}, \dots, t_{n1}, t_{n2}, \dots)$, where t_{ij} is the time for which j th part of the temporary internal state of behavior β_i is stored. Estimates of t_{ij} values can be either learned or supplied by the user. Note that the internal states that we are considering in this paper are minimal information about environment that is needed by behaviors. The internal states do not contain any guidance (e.g. meta-level knowledge) for improving the functionality, except some environmental cues. Thus we expect higher internal states or higher storage times to increase state update computations and thus lead to higher response times and slower functionality.

Since we do not assume any shared memory and the state is completely distributed, it is legal to define the effective internal state as the function g . A chain is more reactive if its effective internal state is lower. We denote the effective internal state of a chain C' by $\Psi_{C'}$.

- **Task space** - A task is defined as a transition from an initial state of the world to a new state, achieved through a temporal chain of behaviors. Thus the chain $\{\beta_1 : \beta_2 : \beta_3\}$ fulfills a task that is different from $\{\beta_1 : \beta_2\}$. The set of all possible temporal chains of behaviors from B is defined as the greatest potential task space, denoted by $\tau_G(B)$.

- **Chain Library** - A chain library, denoted by $S(B)$, is a set of chains composed from the behaviors in B . Note that $S(B)$ is much smaller in size than $\tau_G(B)$. Each chain from $S(B)$ fulfills some task τ_i , defined as $\langle I_i, G_i \rangle$, I_i, G_i being the corresponding initial and goal states. The description of the initial state is complete and that of the goal state is partial. I_i must contain all predicates necessary for the execution of the task fulfilling chains to begin. The initial and goal states are purely conjunctive. These chains from the library are meant for reuse. Each chain can be modified to solve one or more new tasks τ'_i defined as $\langle I'_i, G'_i \rangle$, that a user may assign to the robot.

- **Chain Repairs** - To fulfill a new task, given chain may be modified in two ways - adding a behavior to the chain and removing an existing behavior. The change in the order of behaviors can be described in terms of two addition and two deletion operations. We also refer to these modification operations as *repairs*. The total number of chain repairs is the sum of the number of additions and deletions of behaviors.

- **Chain Robustness** - The amount of modification made to a chain to fulfill a new task can be considered as a measure of its robustness. A chain that requires more modifications as the tasks to be fulfilled change can be viewed as *brittle* and thus less robust. We define a chain to be (i, j) robust if it can fulfill at least j tasks with at the most i repairs. This metric of chain robustness is based on the ease of reusability of a chain. Thus the definition of robustness subsumes the ability to fulfill new tasks with some modification.

5 Chain Synthesis

We discuss how the problems of controlling the reactivity and robustness of behavior chains can be specified and solved.

5.1 Controllable Reactivity

We discuss two important versions of this problem that are general enough to cover the reactivity-related requirements. **(1)** Synthesize a behavior chain of length k or less that not only fulfills the given goal but also has the effective internal state less than some limit γ . An algorithm to solve this problem can be implemented as a search procedure that starts with the initial state and selects behaviors till a task fulfilling chain C' of length k or less is found such that $\Psi_{C'} < \gamma$. This in the worst case, will take $O(|B|^k)$ time, since each behavior in the tree of height k may be chosen after examining $O(|B|)$ options. **(2)** Given a behavior chain C' of length k fulfilling a task such that $\Psi_{C'} > \theta$, modify the chain to C'' such that $\Psi_{C''} < \theta$. Let us assume that the chain is minimal, so no behavior can be removed from the chain to lower $\Psi_{C'}$ without losing the task fulfilling capability. The only option then is to reorder the existing behaviors in the chain C' to derive C'' that satisfies the inequality. Though a brute force algorithm will visit $k!$ orderings in the worst case, a number of strategies/heuristics that we propose in 6.1 can be used to restrict the computation in both **(1)** and **(2)** to be practical.

5.2 Controllable Robustness

The problem of controlling robustness can be stated in a variety of ways. We discuss two important versions here. **(1)** Given a set of tasks Υ , synthesize a set (library) $S(B)$ of p chains such that each chain is (i, j) robust, such that for each task $\tau \in \Upsilon$, there exists a chain C in the set $S(B)$, such that C can be

repaired with at the most i modifications to fulfill τ , and C can fulfill at least j such tasks with such repair. A naive algorithm to solve this problem is to compute all possible minimal chains to fulfill each task and then process the chains by adding or removing behaviors to achieve the (i, j) robustness and then store these chains. This algorithm will explore a combinatorially large search space. **(2)** Given an (i, j) robust chain, derive an $(i + i', j + j')$ robust chain from it, such that $i' < 0, (i + i') > 0, j' > 0$ (this makes an existing chain more robust). A brute force algorithm for this problem will be inefficient. To deal with the computational burden in **(1)** and **(2)**, we suggest several strategies/heuristics in 6.1.

5.3 Repairing the Robust Chains

In 5.1 and 5.2, we discussed the problems of coming up with chains that meet the specified reactivity and robustness criteria. This requires the chains to be synthesized from scratch. The resulting chains are then stored in the chain library. In this subsection, we discuss an incremental, rather than the from-scratch problem. In the incremental problem, modifications to the chains already in the library are carried out, to meet the new robustness and reactivity constraints.

Let us examine the complexity of applying a fixed number of repairs (k) to a given chain of length m . Let the k repairs contain k_1 behavior additions and k_2 deletions, $(k_1 + k_2) = k$. As the insertions and deletions take place, the length of the chain varies, changing the possible number of deletions and possible places for additions. The maximum length of the modified chain will be $(k_1 + m)$. In the worst case, no portion of the reused chain will be applicable and an entirely new chain of length $(k_1 + m)$ will have to be synthesized from scratch. Since each behavior can be chosen in $O(|B|)$ ways, the time complexity of this from-scratch synthesis will be $O(|B|^{(k_1 + m)})$.

The primary motivation for repairing the chains is to reuse them to fulfill new tasks. However this repair should not excessively increase the effective internal state. The computation of effective internal state can be applied to each chain explored in the space of repairs, to control the potential increase in Ψ . Thus the complexity analysis above also applies to the general case where a robust chain is repaired to fulfill a new task and also meet an internal state constraint.

6 Discussion

The ideas presented here can be implemented via a new hybrid architecture for autonomous robots. This architecture will have three components - the behavior space, the chain library and the chain repair mechanism. A user can supply a list of tasks to a robot for it to fulfill autonomously.

6.1 Computational Considerations

We discuss several strategies/heuristics to reduce the computations in repairing the chains to fulfill tasks and the computations in modifying the chains to reduce the effective internal state. These strategies can be applied together to synthesize chains that are adequately reactive and robust. Some of these strategies can also be used to efficiently generate chains that are stored in the chain library for future use, the problem that we discussed in 5.1 and 5.2.

6.1.1 Reducing Chain Repair Computation

(a) Restricting the Repair Nature - We can either allow behavior modules to be only deleted from the chain or only added, but not the both. This does offer a significant reduction in the space of possible repairs. **(b) Restricting the Repair Location** - We can restrict the behavior modules to be inserted at and removed from only certain places rather than all possible places. This offers a significant reduction in the repair space. **(c) Relevance Analysis** - One can use polynomial time pre-processing techniques like the “operator graphs” of [26] that compute the set of actions relevant to the given problem using the information about the initial state, goal state and the stimuli and consequences of the actions. One can similarly construct a behavior graph and use it to select an appropriate chain from the library for modification and then remove irrelevant behaviors from this chain and add only the relevant behaviors if they are missing. **(d) Using Macro-behaviors** - A behavior chain C that fulfills a task τ can be viewed as a macro-behavior. Thus to fulfill the task τ , one need not synthesize a behavior chain from scratch, but one can just retrieve C and execute it. A frequently useful behavior chain can be stored as a sequence of macro-behaviors. This restricts the locations of repair. For example, the chain $\{\beta_1 : \beta_5 : \beta_3 : \beta_4 : \beta_7\}$ can be also stored as $\{C_1 : C_2 : \beta_7\}$, where $C_1 = \{\beta_1 : \beta_5\}$, $C_2 = \{\beta_3 : \beta_4\}$ are the macros. Similarly the new modules to be added to a behavior chain can be restricted to be macros rather than singleton behaviors. **(e) Advice from the Domain Expert** - Just like domain experts provide guidance to planners in the form of hierarchical problem decomposition strategies that are captured in the reduction schemas [17], experts can also provide additional constraints on the process of repairing the chains. **(f) Division of Functionality** - By bounding the scope of environment and the tasks that a robot is responsible for, one can make sure that the size of the chain library and the repair computations are reasonable. **(g) Compaction of the library** The behavior chains may be replaced by a fewer number of partial orders on the behaviors. Similarly

the set of tasks that repaired versions of different chains can solve should ideally be non-overlapping, so that the size of the library is not huge. Certain tasks may be reliably fulfilled just because the behaviors get chained though the world in a linear fashion, without any possibility of branching. Then, only those tasks that need chains that cannot be reliably composed through the world should be solved by repairing the chains. Thus the chain library should have chains for only the latter type of tasks. **(h) Interleaving Execution and Repair** It is not necessary to retrieve and repair a chain immediately after a task is chosen for fulfillment. A robot can execute the applicable behaviors (provided the initial state in the task specification is true) and retrieve and repair a chain only if no progress is made in the execution. Some tasks may be fulfilled without any chain repair, thus the repair effort can be avoided.

6.1.2 Reducing Computations in Control of Reactivity

We suggest several heuristics to reduce the computations in reducing the effective internal state of a chain below. Most of these heuristics order all or selected behaviors in the chain to reduce the effective internal state. **(a) High Internal State First** The behaviors which have a higher value of the temporary internal state component δ_{it} should be ordered to occur as early in the chain as possible, with the constraint that this reordering should not affect the task fulfilling capability of the chain. **(b) State Externalization** Internal state can be externalized by putting markers in an environment. For example, if it is desired that all the recently bought groceries should be kept in the cupboard or refrigerator, except some that are required to cook dinner, their location can be stored as an internal state in the stimulus of the behavior for picking up the groceries. But one can also externalize this state by keeping the groceries to be used on a marker like red paper and modify the stimulus of the behavior by replacing the internal state by the description of the marker. **(c) Thresholding** Behaviors β_i whose internal state $f(\delta_{it}, \delta_{ip})$ exceeds a certain threshold θ may not be included in the chain. **(d) Selective Ordering** Instead of ordering all the behaviors in a chain based on their internal state, only a selected ones may be ordered. **(e) Model of Causality** The stimulus s_i of a behavior β_i in a chain is true either at the beginning of the chain or it is made true by behaviors like β_j in the chain such that $\beta_j \prec \beta_i$, where \prec denotes temporal partial order. Though such causality is never explicitly represented in behavior-based approach, it is responsible for the formation of chains. These causality constraints can be used to prune many of the reorderings that may be tried while reducing the effective inter-

nal state. **(f) Exploiting Concurrency** A behavior chain is a sequence (the behaviors are totally ordered). However there may be non-interfering behaviors whose stimuli are true at the same time, making the concurrent execution of these behaviors possible. Thus a chain can be non-linearized for the purpose of execution to reduce the effective internal states, since the non-linearization of the chain decreases the idle times of the concurrent behaviors.

6.2 Connections with AI Planning

The idea of storing, retrieving and repairing the behavior chains is similar to reusing plans. The stimulus, consequence definition of a behavior is similar to the pre-condition, post-condition (effects) definition of an action or operator in planning. Ginsberg [14] proposes that to alleviate the complexity of plan synthesis, one can use a planner that generates plans that are approximately correct. Our chain library can be considered as a set of approximate solutions that are repaired to get the accurate ones. Our view of the robustness of a chain is similar to the notion of robustness of the solution of a constraint satisfaction problem in [15], where a solution is considered to be more robust if it can be modified with lower effort to handle more constraints. Both Hanks & Weld [16] and Kambhampati & Hendler [17] point out that the efficiency of reusing plans to solve new problems depends crucially on the amount of modification applied to the original plan. Thus it is very important that the reused plan be as close to the solution plan as possible. Thus heuristics for selecting good candidates for reuse are important. These arguments apply to our strategy of repairing behavior chains from the chain library as well and the relevance analysis we suggest in 6.1.1 is this kind of heuristic. The computational complexity-based argument in [22] shows that in the worst case, reusing a plan can be harder than planning from scratch. This argument applies to chain modification too. A combination of various remedies we suggested in 6.1 can however reduce the computations to be practical.

7 Conclusion

We reviewed a number of architectures of autonomous robots and the conventional interpretations of reactivity and robustness. Arguing that there are no methods for designing and synthesizing chains whose reactivity and robustness can be controlled, we formalized these notions. We discussed how our metrics can be used to synthesize and modify behavior chains with a control over the reactivity and robustness properties. We suggested several strategies for reducing the computational effort involved in the synthesis and modification of these chains. With this reduction, it is possible to reduce the times for task completion. We

showed how our metrics and the methods to control them can be incorporated into an architecture for autonomous robots that need to exhibit a robust and reactive behavior.

Our architecture has potential applications in perceptually challenging environments where object recognition times are high and thus distributed internal states need to be maintained and updated to avoid repetition of the recognition computations. It can also be used in the scenarios where the types and magnitudes of changes in the requirements of a user or changes in an environment are bounded, since these can be handled by repairing the chains.

References

- [1] **Philip E. Agre**, Computational research on interaction and agency, *Artificial intelligence* 72, 1995, 1-52.
- [2] **Anderson, T. L. and Donath, M.** 1990. Animal Behavior As A Paradigm For Developing Robot Autonomy, *Robotics and Autonomous Systems*, 6(1 & 2): 145-168.
- [3] **Arkin, R. C.**, Behavior-Based Robot Navigation for Extended Domains, *Adaptive Behavior* 1(2), 1992, 201-225.
- [4] **Ronald C. Arkin and Tucker Balch**, AuRA: Principles and Practice in Review, *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, No. 2, 1997, 175-189.
- [5] **R. Peter Bonasso, H. James Antonisse and Marc G. Slack**, A reactive robot system for find and fetch tasks in an outdoor environment, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1992, 801-808.
- [6] **Brooks R. A.**, Intelligence without representation, *Artificial intelligence* 47, 1991, 139-159.
- [7] **David Chapman**, Penguins can make cake, *AI magazine*, Winter 1989, 45-50.
- [8] **Patrick Colgan**, editor, *Quantitative ethology*, John Wiley & Sons, 1978.
- [9] **Connell, J.**, *Minimalist mobile robotics, A colony style architecture for an artificial creature*, Academic press Inc., 1990.
- [10] **Fikes R.E & Nilsson N.J.** STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* 2: 251-288, 1971.
- [11] **R. James Firby**, An investigation into reactive planning in complex domains, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1987, 202-206.
- [12] **R. James Firby and Reid Simmons**, Mobile Robots II: Architectures for reaction and deliberation, Tutorial presented at the National Conference on Artificial Intelligence (AAAI), 1993.
- [13] **Michael P. Georgeff and Amy L. Lansky**, Reactive reasoning and planning, *Readings in planning*, Morgan Kaufmann, 1990, 729-734.
- [14] **Matthew L. Ginsberg**, Approximate planning, *Artificial Intelligence* 76, 1995, 89-123.
- [15] **Matthew Ginsberg, Andrew Parkes and Amitabha Roy**, Supermodels and robustness, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1998.
- [16] **Steve Hanks and Daniel S. Weld**, A domain-independent algorithm for plan adaptation, *Journal of Artificial Intelligence Research* 2, 1995, 319-360.
- [17] **Subbarao Kambhampati and James A. Hendler**, A validation-structure-based theory of plan modification and reuse, *Artificial Intelligence* 55, 1992, 193-258.
- [18] **David Kirsh**, Today the earwig, tomorrow man? *Artificial intelligence* 47, 1991, 161-184.
- [19] **C. Ronald Kube and Hong Zhang**. Collective Robotics: From Social Insects To Robots. *Adaptive Behavior*, Vol. 2, No. 2, 189-218, 1994.
- [20] **Maja Mataric, Martin Nilsson and Kristian Simsarian**, Cooperative multi-robot box-pushing, Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS) , 1995, 556-561.
- [21] **David Miller, Rajiv Desai, Erann Gat, Robert Ivlev and John Loch**, Reactive navigation through rough terrain: Experimental results, Proceedings of the National Conference on Artificial Intelligence (AAAI), 1992, 823-828.
- [22] **Bernhard Nebel and Jana Koehler**, Plan reuse versus plan generation: a theoretical and empirical analysis, *Artificial Intelligence* 76, 1995, 427-454.
- [23] **Stanley J. Rosenschein and Leslie Kaelbling**, A situated view of representation and control, *Artificial Intelligence* 73, 1995, 149-173.
- [24] **Alessandro Saffiotti, Kurt Konolige and Enrique Ruspini**, A multivalued logic approach to integrating planning control, *Artificial intelligence* 76, 1995, 481-526.
- [25] **Reid Simmons**, Structured control for autonomous robots, *IEEE transactions on robotics and automation*, Feb. 1994.
- [26] **David Smith and Mark Peot**, Suspending recursion in causal link planning, Proceedings of the international conference on Artificial Intelligence Planning Systems (AIPS), 1996.