

---

# Convergent Reinforcement Learning for Hierarchical Reactive Plans

---

Daniel Shapiro

Ross Shachter

Department of Management Science and Engineering, Stanford University, Stanford, CA 94305 USA

DGS@STANFORD.EDU

ROSS.SHACHTER@STANFORD.EDU

## Abstract

Hierarchical reinforcement learning techniques operate on structured plans. Although structured representations add expressive power to Markov Decision Processes (MDPs), current approaches impose constraints that force the associated convergence proofs to depend upon a subroutine-style execution model that restricts adaptive response. We develop an alternate approach to convergent learning that employs hierarchical plans with an interruptible (reactive) execution model. We prove this format reduces to an MDP, and thus that we can implement any algorithm that converges on MDPs directly on reactive plans. We introduce one such algorithm, called Sharsha(0) (an acronym for state *hierarchy*, action, state *hierarchy*, action) that simultaneously finds the Q-values for subplans and the optimal hierarchical policy via an on-line exploration of a single, infinitely long execution sequence.

## 1. Introduction

Many reinforcement learning algorithms represent behavior with Markov Decision Processes (MDPs). Although well understood, MDPs suffer from a problem of scale since the complexity of finding a solution grows so rapidly with the size of the domain. Research in hierarchical reinforcement learning addresses this concern by incorporating domain knowledge into the statement of the learning problem. While this strategy reduces search, it leads to an interesting tension. On one hand, the desire to solve complex problems argues for the use of powerful representations. On the other, the desire for convergent learning suggests a simpler format that offers a clear mapping into the MDP model, since MDPs provide the conceptual framework for developing algorithms and the mathematical properties underlying convergence proofs.

Research in hierarchical reinforcement learning addresses this tension by developing carefully constrained representations. For example, Sutton, Precup, and Singh

(1998) define macro-actions that fix behavior over a region of the state space and learn an optimal policy that switches among these options. Parr and Russell (1998) define a hierarchy of non-deterministic finite-state machines and employ learning to extract an optimal deterministic controller. Dietterich (1998, 2000) points out that these approaches require a full description of system state and shows that MAXQ converges under appropriate state abstractions. These designs all share a common thread: they provide expressive power through hierarchical plans, and show convergence by mapping those formats into the MDP or semi-MDP framework. However, the added structure has a price: it causes the convergence proofs to depend upon a subroutine-style execution model that limits adaptive response. (Although the designs all support interruptible (reactive) execution, its use invalidates the convergence theorems.)

This paper develops an alternate path to convergent reinforcement learning that builds on purely reactive plans as opposed to subroutines with temporal extent. Reactive plans repeatedly execute a method of mapping observations into a representation of intent. Extremely reactive plans (the kind investigated in this paper) make no commitments about control flow; they simply select and apply a series of situation-relevant actions that accomplish the desired intent over time. This format has several advantages relative to the use of subroutines for reinforcement learning. In particular, we will show that reactive plans transform cleanly into MDPs, since both models ignore previous state. This frees the resulting mathematical development from any concern with termination criteria and execution intervals. The reactive format also supports a powerful vocabulary for representing intent. It natively includes the ability to act on partial information (state abstraction), and it can incorporate goals, constrained choices (similar to partial policies), mandatory behavior (similar to options), and hierarchical problem decomposition. Universal plans (Schoppers, 1987) and teleoreactive trees (Nilsson, 1994) are cases in point. We adapt and extend those systems in the direction of a practical, reactive programming language (Shapiro, 1999).

The following sections develop the relation between reactive plans and reinforcement learning techniques. First, we show that reactive plans are reducible to MDPs. This establishes that we can learn optimal policies for reactive plans via any algorithm that converges on MDPs. Next, we show that we can avoid explicit transforms among representations by re-implementing a well-known algorithm, Sarsa(0), on reactive plans. We conclude by presenting a hierarchical learning algorithm specifically adapted to the layered representation of intent inherent in reactive plans, and we analyze its convergence using the correspondence between reactive plans and MDPs.

## 2. Reinforcement learning with MDPs

Reinforcement learning addresses an optimal control problem; it generates a control policy that causes a stochastic, dynamic system to maximize an objective function over time. Most algorithms are specialized, however, to treat system dynamics as an unknown. As a result, reinforcement learning algorithms explore the space of possible control sequences and extract a direct map from situation and action to anticipated reward.

If the application supports suitable Markov assumptions, the reinforcement learning problem reduces to the task of solving an MDP with an unknown state transition function. An MDP identifies a set of mutually exclusive and collectively exhaustive states  $S = 1, 2, \dots, N$  and a discrete set of available actions,  $A$ , where some subset of  $A$  is relevant in each state. We denote the probability of making a transition from a state  $s$  to state  $s'$  on taking action  $a$  by  $P_{ss'}^a$ , and the random payoff for taking  $a$  in  $s$  by  $r(s, a)$ . The Markov assumption requires that  $P_{ss'}^a$  and  $r(s, a)$  be independent of all previous states and action.

Figure 1 illustrates the states and actions in an MDP for piloting a bumper car at an amusement park. We assume the arena contains two cars, one piloted by the agent, and the other by a driver who employs a fixed but unknown situation-action map utilizing the same primitive actions. We label states by the extant environmental conditions, specifically the distance to the other car and the emotional state of its driver. (These labels have no meaning to the Markov process.) We associate the full set of feasible actions with each state, following a common practice in framing reinforcement learning problems.

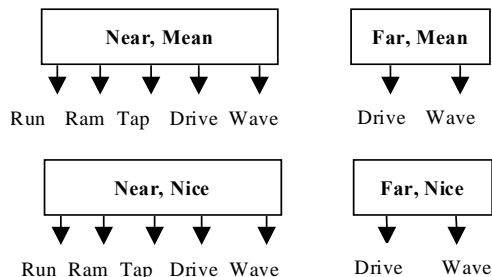


Figure 1. States and actions in bumper cars.

For example, Drive laps and Wave are feasible in the states {Far, Mean} and {Far, Nice}, while Run, Ram, and Tap demand proximity to the other vehicle. We impose a Markov structure on this domain by assuming that (1) the probability the agent makes any state transition,  $P_{ss'}^a$ , and (2) the reward associated with all state-action pairs are independent of preceding states and actions.

The agent's payoff is an unknown, random function of its state-action pair. For the sake of argument, we assume the agent gets reward for hitting the other car but absorbs a penalty for being hit. That is,

$$r(s, a) \equiv \begin{cases} 1 & \text{if collision and } a = \text{Ram} \\ .5 & \text{if collision and } a = \text{Tap} \\ -1 & \text{if collision and } a \neq \text{Ram, Tap} \\ 0 & \text{otherwise} \end{cases}$$

The optimal policy in bumper cars is a mapping from each state to a single driving action. These actions collectively interact with the environment and the other driver's control program to maximize the agent's future discounted reward stream. Many learning algorithms can acquire this policy. For example, Sarsa(0) updates estimates (Q-values) for the future reward stream of a state-action pair using the observed reward and the estimate associated with its successor state via the rule:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r(s, a) + \gamma Q(s', a')]$$

where  $0 < \alpha < 1$  is the learning rate and  $0 < \gamma < 1$  is a discount factor. Singh et al. (in press) have recently shown that Sarsa(0) converges to the correct estimates and the optimal policy (the one obtained by selecting the action with the highest Q value) under reasonable procedures for exploring the control space and damping the learning rate.

## 3. Hierarchical reactive plans

A reactive plan is a program for selecting action that emphasizes situated response. A *hierarchical reactive plan* (HRP) structures that program in terms of abstract and specific intent (Nilsson, 1994; Schoppers, 1987). HRPs have application in physical domains (Brooks, 1986; Gat, 1992; Schoppers, 1995). While they have many possible representations, we employ an interleaved set of queries and decision opportunities that decision theorists would call a *decision tree*.

Figure 2 gives an example from the bumper car domain. Here, circles represent tests on the environment, squares are decisions, and the outgoing arcs to the right of decisions denote actions. Primitive actions are printed in bold, e.g., to Run or Ram; they form the leaves of the tree. We represent a non-primitive action, or *subplan*, by a subtree (e.g., to Measure the other driver's attitude before selecting action). An HRP can contain any number of environmental tests and decisions, interleaved in any order. However, the leaf nodes must represent decisions.

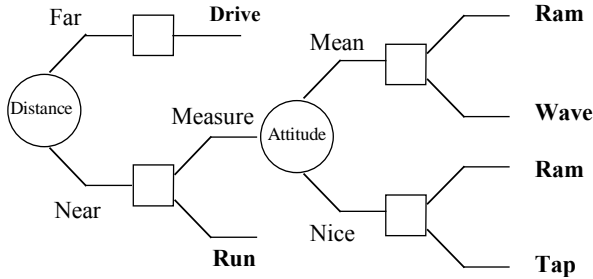


Figure 2. A hierarchical reactive plan for bumper cars.

Hierarchical reactive plans employ an interpreter to evaluate such trees. We consider an iterative design that examines the plan in a top-down fashion, starting at the leftmost node every execution cycle. The interpreter resolves environmental queries at test nodes and transits the appropriate outgoing arc, then selects an alternative at the subsequent decision node. On reaching a leaf node, it applies the associated primitive action and immediately returns to the top node for the next iteration. The primitives are processes that can be executed one step; they need not run to completion. The HRP interpreter implements a reactive control policy that allows the locus of execution to shift in response to environmental changes from one subplan to any another on successive execution cycles. For example, the agent might Ram a nice opponent on cycle one (after choosing to Measure its attitude), Wave at the suddenly mean opponent on cycle two, and then abandon those strategies in favor of Driving laps as the other car moves out of range. Thus, HRPs define a hierarchy of intent, but not a hierarchy of temporal extent as is common in reinforcement learning.

HRPs provide a simple programming language for expressing reactive strategies. Where MDPs typically identify the full set of feasible behaviors as input to a learning method, HRPs encourage system designers to focus on a subset of worthwhile options that are likely to generate reward. This requires (or exploits) domain knowledge about the behavior of the environment and the structure of the reward function. (We assume the same reward function shown in the previous section.) Figure 2 illustrates an idiosyncratic strategy that forces the agent to Drive laps whenever the other car is far away, but lets the agent measure and respond to the other driver's attitude if the car is near. Perhaps mean-seeming people can be mollified (made nice) by Waving, but if not, it is worth points to Ram them before they Ram you.

#### 4. The equivalence between HRPs and MDPs

While HRPs and MDPs define methods of reacting to current situations, they differ in significant ways. In particular, HRPs expand the state diagnosis process that is implicit in MDPs, they act on the basis of partial information, and they can select many subplans before arriving at a primitive action. Thus, it is not clear how

HRPs embody the concepts of state and state-action pairs that are so central to MDP-based convergence proofs.

We show that HRPs are equivalent to MDPs under several plausible statements of the Markov assumption. That is, we identify conditions in the application domain that must hold before we can reduce HRPs to MDPs. We begin with the following Lemma:

*Lemma: Every MDP is equivalent to an HRP.*

We can transform an MDP into an HRP via a syntactic manipulation: take any MDP, preserve the transition probabilities, rewards, and decision alternatives, then add a state-diagnosis query at the head of the situation-action map. The resulting HRP encodes the MDP as an iteratively interpreted diagram, as illustrated in Figure 3.

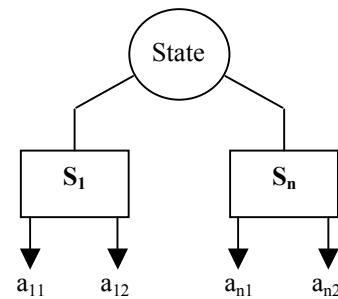


Figure 3. The HRP for an arbitrary MDP.

We need some notation in order to show that every HRP is equivalent to some MDP. Let  $F(t) = [f_1(t), \dots, f_n(t)]^T$  be a partial feature vector measured while traversing the HRP in interpreter cycle  $t$ . The number of elements in this feature vector grows as the interpreter progresses from the root towards the leaves of the tree. For convenience, let  $F_t$  be the largest (and final) feature vector encountered on cycle  $t$ . It contains the most informed picture of state garnered that cycle. Let  $a_t$  be the primitive action chosen at cycle  $t$ , after observing  $F_t$ .

The key issue in reducing HRPs to MDPs is to clarify the notion of HRP state. We do this by introducing two alternate modeling assumptions:

Weak Markov assumption:  $F(t)$  is conditionally independent of  $F_{t-k}, a_{t-k} \forall k > 1$ , given  $F_{t-1}, a_{t-1}$ .

Strong Markov assumption:  $f_n(t)$  is conditionally independent of  $F_{t-k}, a_{t-k} \forall k \geq 1$ , given  $f_1(t), \dots, f_{n-1}(t)$ .

We will require that the reward,  $r(t)$ , is also conditionally independent of  $F_{t-k}, a_{t-k} \forall k > 1$ , given  $F_{t-1}, a_{t-1}$ .

The Weak Markov assumption maps onto our understanding of MDPs if we substitute the word *observations* for *state*. In this phrasing, past history has no impact on current observations given we know the immediately preceding observations and action. Consider the HRP for bumper cars in Figure 2. Here, the Weak

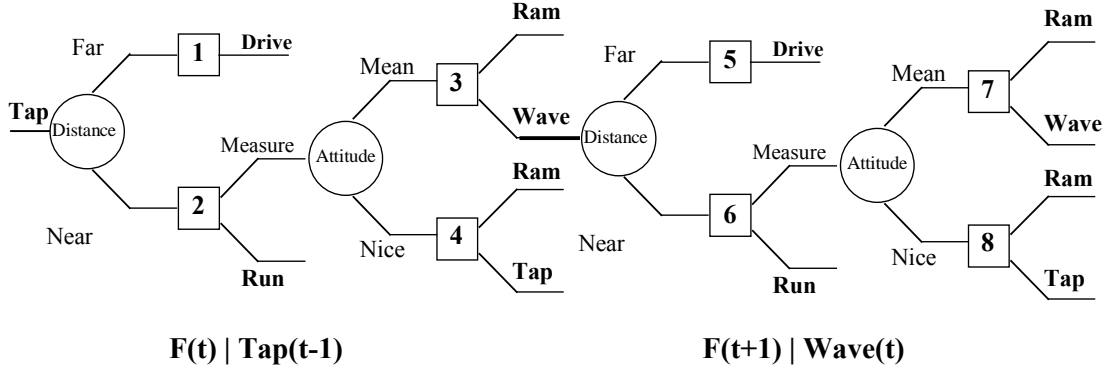


Figure 4: The graph obtained by unrolling an HRP in time.

Markov assumption implies that one Tap is as good as twenty in terms of its potential to transform the other driver's attitude from nice into mean. Similarly, a more recent Ram overwrites the effects of any preceding Tap. The Weak Markov assumption allows (but does not require) current observations to depend upon events at cycle  $t-1$ . This lets  $p(\text{near}(t) | \text{far}(t-1), \text{Drive}(t-1))$  differ from  $p(\text{near}(t) | \text{near}(t-1), \text{Run}(t-1))$ , as one might expect.

The Strong Markov assumption implies the Weak Markov assumption because it requires each observation at cycle  $t$  to be independent of *all* observation and action in previous cycles. This corresponds to the typical application of reactive plans, as situation-action maps devoid of previous context. Note that we can construct an HRP that obeys the strong assumption when the weak assumption would seem to apply by adding observations to the top of the tree. Thus, we can 'observe' the previous action at the start of the current time cycle, or the previous feature vector and action both. The cost is an exponential growth in the size of the HRP.

Now that we have defined MDPs, HRPs, and two Markov assumptions, we move to the central theorem.

**Theorem 1:** *Any hierarchical reactive plan satisfying the Weak Markov assumption can be reduced to an MDP.*

**Proof:** The proof is by construction of the transition and payoff functions that define an MDP under a suitable definition of system state. Figure 4 provides a graphical aid that expands the HRP in time. More exactly, this diagram appends two copies of the HRP, where each is specialized by the instance of the primitive action taken on the previous iteration. So, assuming  $a_{t-1} = \text{Tap}$ , we imagine the HRP interpreter retrieving the Wave action from the left-hand diagram at time  $t$ , and proceeding to the right-hand diagram at time  $t+1$ . Note that the conditioning information required by the Weak Markov accumulates from left to right in this graph. We construct an MDP from an HRP by associating states with decision nodes and transition probabilities with environmental

tests. We extract transition probabilities after numbering decision nodes in any unique fashion. For example:

$$P_{16}^a = 0$$

$$P_{36}^{\text{wave}} = p(\text{near} | \text{wave}(t))$$

$$P_{67}^{\text{measure}} = p(\text{mean} | \text{near}, \text{wave}(t))$$

Here, the probability of transiting from any state to another that is not its successor is zero, while the probability for an allowable transition is conditioned on the current feature vector and preceding action. We extract the matrix of payoffs,  $R$ , with a similar logic:

$$R_{16}^a = \text{undefined}$$

$$R_{36}^{\text{wave}} = E[r(\text{wave} | \text{near}, \text{mean})]$$

$$R_{67}^{\text{measure}} = 0$$

The reward for an unallowable transition is undefined, while the expected reward for a primitive action is solely conditioned by the features measured that interpreter cycle. The reward for any non-primitive action is zero.

We obtain the transition and payoff matrices for the general case by employing a more careful policy to name decision nodes (states). Assume the Weak Markov assumption holds. Let the set of states,  $S$ , be all tuples of the form  $\{F(t), F_{t-1}, a_{t-1}\}$ , where  $F(t)$  is the partial feature vector measured while traversing the HRP in interpreter cycle  $t$ ,  $F_{t-1}$  is the full feature vector observed in cycle  $t-1$ , and  $a_{t-1}$  identifies the specific instance of the primitive action taken in cycle  $t-1$ . Each decision node in Figure 4 corresponds to one  $\{F(t), F_{t-1}, a_{t-1}\}$  tuple, and the set of possible tuples identifies every unique decision context the interpreter can encounter.

The probability of a transition from state  $s$  to state  $s'$  is:

$$P_{ss'}^a \equiv \begin{cases} P_{ss'}^a & \text{if } s' \in \Gamma(s, a) \\ 0 & \text{otherwise} \end{cases}$$

where  $\Gamma$  is the successor operator, defined as in Figure 4. Note that  $P_{ss'}^a$  is nothing more than the probability associated with an environmental query in the HRP, since

the definition of state includes all of the potential conditioning information permitted by the Weak Markov assumption. The payoff function follows by analogy:

$$R_{ss'}^a \equiv \begin{cases} R_{ss'}^a & \text{if } a \text{ is primitive and } s' \in \Pi(s,a) \\ 0 & \text{otherwise} \end{cases}$$

The states  $S$ , the transition probabilities  $P_{ss'}^a$ , and the payoff function  $R_{ss'}^a$  collectively define the MDP entailed by any given hierarchical reactive plan. The Strong Markov assumption leads to a similar MDP with less information embedded in the definition of state. ♦

## 5. Sarsa(0) on a Hierarchical Reactive Plan

Given that hierarchical reactive plans are reducible to MDPs, we can implement any MDP-based algorithm on the HRP data structure. We illustrate this process using Sarsa(0), an on-line single-step temporal difference method commonly discussed in texts that cover reinforcement learning (Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998). Sarsa is an acronym for “state, action, reward, state, action”. It associates Q-values with state-action pairs, selects action according to some probabilistic rule that permits exploration, and performs updates using two successive state-action pairs via the rule:  $Q(s,a) \leftarrow (1 - \alpha)Q(s,a) + \alpha[r(s,a) + \gamma Q(s',a')]$ .

Singh et al. (in press) show that Sarsa(0) converges given an exact (tabular) representation of Q-values, and a “restricted, rank-based, randomized” action selection rule (RRR) that picks alternatives with a probability that descends in the rank-order of their estimated Q-values. We assume other technical assumptions hold (regarding learning rates, bounded rewards, and the like). Given this context, the authors show that (1) Sarsa(0) converges to the optimal (greedy) policy if the exploration rate decays to zero, and (2) Sarsa(0) converges to an optimal policy including exploration if the exploration rate remains positive. We build on both results.

We implement Sarsa~ (Sarsa(0) applied to an HRP) by merging the RRR selection and Sarsa update rules into the HRP interpreter. This requires a method of associating Q-values with state-action pairs that applies the full definition of HRP state per the Weak Markov assumption. The tree structure (Figure 2) provides an advantage here, since the locus of execution in the HRP identifies the feature vector observed that cycle. This allows us to encode state as  $(a,p)$  pairs, where  $a$ , implying  $F(t)$ , represents a primitive action or subplan (specifically an arc for an alternative in the tree), and  $p$  (implying  $F_{t-1}$ ) is the arc representing the primitive executed on the previous time cycle. The algorithm is in Table 1.

Here, the function *RRR* returns an alternative (an arc) chosen by rank-ordered selection, and *LookupQvalues* returns a set of  $(Qvalue, alternative)$  pairs. Since Sarsa(0)

converges under RRR on MDPs, and hierarchical reactive plans reduce to MDPs, we have the following theorem:

**Theorem 2:** *Given the Weak Markov assumption, Sarsa~ converges on an HRP.*

In particular, Sarsa~ converges, both with and without persistent exploration, to the same policies and Q-values obtained by Sarsa(0) applied to the corresponding MDP.

Table 1: Sarsa~: Sarsa(0) implemented on an HRP

```

Initialize  $Q(a, p)$  arbitrarily  $\forall a, p$ 
Initialize  $p$  to some instance of a leaf action
 $a \leftarrow p$ 
 $N \leftarrow$  top node of HRP
Repeat (for each step of episode)
  Take action  $a$ , measure reward  $r$ 

  Case  $N$ 
     $N$  is a Query node:
      Resolve the query,  $N \leftarrow$  next Node down corresponding ar
     $N$  is a Decision node
       $a' \leftarrow RRR(LookupQvalue\{alternative\}(N), p)$ 
       $Q(a, p) \leftarrow (1 - \alpha)Q(a, p) + \alpha[r + \gamma Q(a', p)]$ 
      if  $a'$  is primitive  $N \leftarrow$  top node,  $p \leftarrow a'$ 
      else  $N \leftarrow$  next Node down  $a'$ 
       $a \leftarrow a'$ 

```

While Sarsa~ demonstrates that we can define convergent reinforcement learning algorithms for hierarchical reactive plans, the next section introduces an algorithm specifically designed to exploit the benefits of hierarchy for complex reactive response tasks.

## 6. Sharsha(0)

Sharsha(0) (an acronym for state-*hierarchy*, action, state-*hierarchy*, action) is an on-line algorithm inspired by the desire to apply reinforcement learning to control physical agents. Problems of this kind typically involve a high-dimensional concept of state, a large number of potential actions, and unexpected changes to world state that confound commitments. Sharsha(0) addresses these concerns by employing hierarchical plans to express domain-specific strategies and the HRP interpreter to provide reactive control. In addition, Sharsha(0) associates all reward with primitive action, as is natural in physical domains. This leads to an algorithm that treats the HRP as an aggregate method for generating primitive actions. We define Sharsha(0) in Table 2.

Table 2: An implementation of Sharsha(0).

$\bar{d} \leftarrow \text{max depth of HRP (in decision nodes)}$   
Initialize  $Q(a, p) \forall a, p$   
Initialize  $p_{t-1}$  to some instance of a leaf action  
 $p_t \leftarrow \text{EvaluateHRP}(p_{t-1})$   
  
Repeat (for each cycle in episode) :  
  Apply  $p_t$ , measure reward  $r$   
   $p_{t+1} \leftarrow \text{EvaluateHRP}(p_t)$   
  For each alternative  $a$  from  $p_t$  to the root :  
     $d \leftarrow \text{depth}(a)$   
     $Q(a, p_{t-1}) \leftarrow (1-\alpha)Q(a, p_{t-1}) + \alpha\gamma^{\bar{d}-d} [r + \gamma^{\bar{d}} Q(p_{t+1}, p_t)]$   
     $p_{t-1} \leftarrow p_t, p_t \leftarrow p_{t+1}$

Sharsha(0) thinks in terms of HRP cycles:  $p_t$  and  $p_{t+1}$  identify the current and next primitive action, while  $p_{t-1}$  captures information about state from the previous cycle required by the Weak Markov assumption. The procedure *EvaluateHRP* generates primitive actions by traversing the HRP from the top down, resolving environmental queries and choosing alternatives at decision nodes via the RRR policy. (The parameter  $p$  completes the definition of state required by the RRR policy to access an appropriate Q-value.) The update operator attributes reward from the current action to all enclosing subplans. Here, the parameter  $\bar{d}$  discounts across interpreter cycles, while  $\bar{d} - d$  discounts within a cycle. This use of  $\bar{d}$  also eliminates any bias the HRP might contain towards short paths to action (which are the only sources of reward in this model). Note that we can reduce the size of the Q-value table by collapsing the states  $(\bar{p}, p_{t-1})$  for all  $p_{t-1}$ , since they have the same immediate reward (by the problem formulation) and the same future (by the Weak Markov assumption). The table only needs one entry,  $Q(\bar{p}, \bullet)$  for each  $\bar{p}$ .

It is not immediately clear that Sharsha(0) should converge, since the distribution for reward at abstract alternatives depends upon subsequent choices (meaning  $r(a, p_{t-1})$  varies during learning). However, Sharsha does reduce to a variant of Sarsa on a corresponding MDP. To see this, consider an HRP whose primitive actions are padded with degenerate decision nodes until all subplans are of even depth, and the entire plan has depth  $\bar{d}$ . This change clearly has no impact on the available choices or policies. Next, we define a version of Sarsa that employs a non-standard multi-step backup procedure requiring it to wait for  $\bar{d} - d + \bar{d}$  counts after selecting an alternative at a decision node before performing an update. This ensures that the update will employ the Q-value obtained from the primitive action found on the next HRP cycle, as well as an appropriately discounted reward. That is:

$$Q(a, p_{t-1}) \leftarrow (1-\alpha)Q(a, p_{t-1}) + \alpha[\gamma^{\bar{d}-d} r + \gamma^{\bar{d}-d+\bar{d}} Q(p_{t+1}, p_t)]$$

exactly as before. So, this variant of Sarsa makes the same decisions as Sharsha(0) at all non-degenerate choice-points, and performs identical updates. Thus, both algorithms learn the same Q-values and policies. We summarize this result in the following theorem:

**Theorem 3:** *Sharsha(0) converges whenever Sarsa under RRR with n-step backup converges.*

While Singh et al. (in press) show Sarsa converges under RRR and a single-step update, the convergence of n-step algorithms for the general control problem has yet to be shown.<sup>1</sup> In practice, it is common to assume n-step methods converge wherever one-step contractions apply.

Sharsha becomes a purely hierarchical application of Sarsa when we eliminate in-period discounting from the update rule. Our implementation of this variant also appears to converge empirically.

## 7. Discussion

We have shown how to combine convergent learning with the expressive power of hierarchical plans and the adaptive response of the reactive execution model. This has several benefits. Relative to the use of flat MDPs, hierarchical representations allow system designers to express partial theories about how to behave, and employ learning to determine the best options from experience. This simultaneously organizes the action selection process, avoids unnecessary observation, and controls the complexity of learning. In addition, the merger of reactive control with learning fosters a natural application in physical domains, where events can interrupt intent at all levels within a plan hierarchy.

Other approaches to hierarchical reinforcement learning share this interest in employing domain knowledge to reduce search. Dayan and Hinton (1993) structure the learning problem by imposing a hierarchy of learning modules (with state abstraction). Kaelbling (1993) structures the solution by forcing the learned policy to pass through an ordered sequence of goals. Sutton, Precup, and Singh (1998) define macro actions (options) that determine system behavior over a region of the domain, and thus restrict learning to the space of possible options from the space of possible actions. Parr and Russell (1998) define partial policies that eliminate choice in some states while restricting choice in others, once again reducing the learning task. Dietterich (2000) defines a hierarchical problem decomposition that combines action on partial information with a segmented reward function. In each of these cases, empirical evidence shows that learning rate increases as added knowledge constrains the size of the learning problem. We have observed the same improvement in reactive plans.

<sup>1</sup> Singh (personal communication, 2000).

The use of reactive plans impacts our formal treatment of reinforcement learning because the plans contain no subroutines. This renders the mathematical development independent of subtask termination criteria (unlike Dietterich, Parr, and Sutton), and lets the convergence proof employ an interruptible execution model. In addition, the reactive interpretation cycle allows us to impose a regular clock on system transitions. This suggests (but does not require) an allegiance to the MDP vs. semi-MDP format that underlies the convergence proofs in each of the above designs. However, we note that our proof is very similar in style to Parr and Russell's, as they map a chosen representation (HAMs) into a known form (semi-MDPs), and then build a convergent algorithm that directly manipulates the more expressive format.

Finally, our use of reactive plans impacts the quality of learned behavior. Learning systems that execute subroutines to completion can only find the best policy consistent with that commitment (no switching mid-stream). In contrast, algorithms that select abstract action on each iteration (called non-hierarchical execution) search a larger space of possible policies. A flat MDP is an extremum in this dimension, since it offers all feasible actions in all possible situations, and learns over the largest space of possible policies. Our work is less extreme (and more constrained): we employ non-hierarchical execution but allow action on partial information about world state. As a result, a hierarchical reactive plan may or may not contain the optimal policy of the flat MDP. If it does, learning will converge to the same policy, but presumably in much less time. Our initial experiments support this proposition.

In our future work, we intend to generalize Sharsha(0) into Sharsha( $\lambda$ ), a multi-step algorithm that includes value approximation. We will also modify HRPs to provide a goal-oriented vocabulary. Given these tools, we hope to produce a programming language for expressing reactive plans that contains a built-in learning algorithm, and then apply it in a variety of applications.

## Acknowledgements

We thank Pat Langley for his interest in merging reactive planning with reinforcement learning, Claude-Nicolas Fiechter for his suggestion to unroll the HRP in time, and Derek Ayers, Mark Pendrith, and Benjamin Van Roy for helpful discussions.

## References

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Belmont, CA: Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, CA: Athena Scientific.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2,1.

Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning. *Advances in Neural Information Processing Systems*, 5, 271-278. San Francisco: Morgan Kaufmann.

Dietterich, T.G. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 118-126). Morgan Kaufmann.

Dietterich, T.G. (2000). State abstraction in MAXQ hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 12. MIT Press.

Gat, E. (1992). Integrating planning and reacting in a heterogenous asynchronous architecture for controlling real-world mobile robots. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 809-815).

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 167-173). Morgan Kaufmann.

Nilsson, N. (1994). Teleoreactive programs for agent control. *Journal of Artificial Intelligence Research*, 1, 139-158.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems*, 10 (pp. 1043-1049). MIT Press.

Schoppers, M. (1987). Universal Plans for reactive robots in unpredictable environments. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 1039-1046). Morgan Kaufmann.

Schoppers, M. (1995). The use of dynamics in an intelligent controller for a space faring rescue robot. *Artificial Intelligence*, 73, 175-230.

Singh, S., Jaakola, T., Littman, M. L., & Szepesvari, C. (in press). Convergence results for single-step on-policy reinforcement learning algorithms. *Machine Learning*.

Sutton, R. S., & Barto, A.G. (1998). *Introduction to reinforcement learning*. Cambridge, MA: MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1998). Intra-option learning about temporally abstract actions. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 556-564). Morgan Kaufmann.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8, 279-292.