Using Abstract Models of Behaviours to Automatically Generate Reinforcement Learning Hierarchies

Malcolm R. K. Ryan MALCOLMR@CSE.UNSW.EDU.AU
School of Computer Science and Engineering, University of New South Wales, Sydney NSW 2052, Australia

Abstract

In this paper we present a hybrid system combining techniques from symbolic planning and reinforcement learning. Planning is used to automatically construct task hierarchies for hierarchical reinforcement learning based on abstract models of the behaviours' purpose, and to perform intelligent termination improvement when an executing behaviour is no longer appropriate. Reinforcement learning is used to produce concrete implementations of abstractly defined behaviours and to learn the best possible choice of behaviour when plans are ambiguous.

Two new hierarchical reinforcement learning algorithms are presented: Planned Hierarchical Semi-Markov Q-Learning (P-HSMQ), a variant of the HSMQ algorithm (Dietterich, 2000b) which uses plan-built task hierarchies, and Teleo-Reactive Q-Learning (TRQ) a more complex algorithm which implements hierarchical reinforcement learning with teleo-reactive execution semantics (Nilsson, 1994). Each algorithm is demonstrated in a simple grid-world domain.

1. Introduction

The main problem that researchers face in reinforcement learning is that of scaling up. The search for a general-purpose solution to arbitrary large learning problems has been largely fruitless. In response to this, some researchers have redirected their attention toward building better special-purpose solutions, systems that deliberately incorporate domain-specific background knowledge in a systematic way to improve learning performance.

Hierarchical reinforcement learning (HRL) is one such technique that is proving quite effective. It implements the familiar intuition that a complex task can be more easily solved if it can be decomposed into a set of simpler tasks. There are several approaches to implementing this (eg: Dietterich, 2000a, Parr & Russell, 1998, Sutton et al., 1999), but most involve the idea of some kind of temporally abstract action, or *behaviour* which has a limited scope (or *applicability space*) and localised goals. Policies are learnt in terms of these abstract behaviours rather than directly in terms of primitive actions.

Simply adding behaviours blindly does not solve the problem. An agent with a diverse repertoire of behaviours with overlapping applicability spaces may have just as much trouble learning a policy as an agent learning a primitive policy directly. Most hierarchical reinforcement learning algorithms are *model-free* – they require no prior model of their behaviours' effects, nor do they build one – and the agent must explore them to learn their effects. Yet behaviours are designed with a *purpose*. This purpose serves as an abstract model which tells us that, from all the applicable behaviours in a particular state, some are *appropriate* and some are not. To save the learning agent from blindly exploring inappropriate behaviours, we need to implement some of this knowledge.

Most existing algorithms achieve this by including some kind of *task hierarchy* which structures the agent's decision process, limiting the set of choices it can make to those that might be productive. Essentially this is a function which maps the agent's state to a set of appropriate behaviours. At present this function is implemented by hand by the trainer. As more ambitious problems are tackled, this is likely to become an increasingly difficult task.

In this paper we provide a means to specify abstract symbolic models of an agent's behaviours and goals. Behaviours are represented by planning operators. Symbolic descriptions are used to allow the trainer to specify behaviours in a high-level language. These operators are then used to automatically construct task-hierarchies through planning. This hybrid of planning and learning allows us to have the best of both worlds – using background knowledge to automatically structure our policies through planning, and reinforcement learning to produce concrete policies for behaviours and optimise choices in the plan.

Furthermore, the plan can also identify when an exe-

cuting behaviour stops being appropriate, and so can be abandoned. We shall show how this can be used to produce more efficient handling of unexpected changes in the world.

2. A Motivating Example

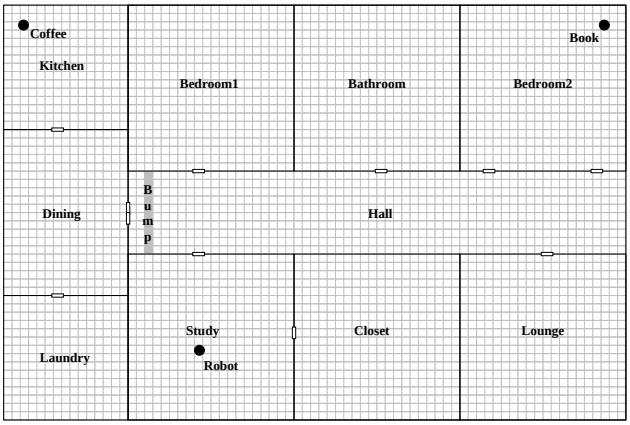

Figure 1. The grid-world

Consider the grid-world illustrated in Figure 1. Imagine that the learning agent is a household robot in a house with the layout shown. Its purpose is to fetch objects from one room to another. It is able to know its location with a precision as shown by the cells of the grid, and its primitive actions enable it to navigate from a cell to any of its eight neighbours, with a small probability of error. (Ignore, for the time being, the section of floor labelled "Bump").

If the robot is in the same cell as an object, it can pick it up and carry it. In the kitchen in the north-west corner of the map is a machine which dispenses a cup of coffee. In the second bedroom there is a book, also indicated on the map. The robot starts at its docking location in the study. Its task is to fetch a cup of coffee and the book and bring them to the lounge. It must learn a policy to achieve this. This is not in itself a difficult task, but it is certainly one that can be made easier by an appropriate set of behaviours.

The obvious behaviours to specify are: Go(`Room1`, `Room2`) which moves the robot between two neighbouring rooms, and Get(`Object`, `Room`) which moves towards and picks up the specified object when the robot is in the same room as it. Rather than implementing these behaviours by hand, we shall define local reward functions for each and learn them as subtasks with a recursively optimal reinforcement learning algorithm (detailed below).

Notice that the agent's repertoire contains a large number of redundant behaviours. Many of the rooms of the house are not worth visiting. Other behaviours are only appropriate in certain situations. For instance, there is no point in the robot entering the lounge room until it has both the book and the coffee in its possession; nor is it worth returning to the bedroom once the book has been collected. An effective task hierarchy should prune these behaviours from the search space to prevent unproductive exploration.

3. Building Task Hierarchies Though Planning

To build a task hierarchy which distinguishes appropriate behaviours from inappropriate, we need to model their intended effects. Such a model is more readily expressed abstractly using high-level language. The language of symbolic planning is ideally suited to this. States, actions and goals are all described symbolically using a language of first-order *fluents*. Fluents describe particular features of the agent's state. In the grid-world, such fluents might be `loc(Object, Location)`, which is true when the `Object` (be it the coffee, the book, or the robot itself) is in the specified `Location` (a room, or `robot` if it is in the robot's possession), and `door(From, To)` which is true if the rooms specified by `From` and `To` are connected.

Using this language the robot's starting state would be described as the conjunction:

$$s \models \texttt{loc(robot, study)} \wedge \texttt{loc(coffee, kitchen)} \wedge \texttt{loc(book, bedroom2)}$$

and its goal would be:

$$G = \texttt{loc(robot, lounge)} \wedge \texttt{loc(coffee, robot)} \wedge \texttt{loc(book, robot)}$$

The Go() and Get() behaviours described above could also be described in this language as *teleo-operators* (Nilsson, 1994). Teleo-operators define the effects of a goal-directed behaviour B in terms of its *pre-image* B.pre and *postcondition* B.post. B.pre and B.post are sets of states described by conjunctions of fluents. If B is initiated in a state satisfying B.pre then it will maintain that condition until it achieves its postcondition B.post. (Unlike earlier models of action, such as STRIPs, teleo-operators do not assume that actions operate instantaneously).

Thus the Go() and Get() behaviours can be expressed as teleo-operators as follows:

Go(Room1, Room2)	
Pre:	loc(robot, Room1) ∧ door(Room1, Room2)
Post:	loc(robot, Room2)

Get(Object, Room)	
Pre:	loc(Object, Room) ∧ loc(robot, Room)
Post:	loc(Object, robot)

Given this information, a symbolic planner can construct a plan which sequences these behaviours to achieve the specified goal. In this paper we shall use *means-ends planning* (Newell & Simon, 1972) for this purpose. This is a simple and well-known technique, and we shall not enter into the details of it here. (More complex modern planning techniques could also be adapted to this purpose but have been avoided for simplicity's sake.) Figure 2 shows part of the plan produced to solve the grid-world problem. Two modifications have been made to the standard means-ends algorithm: Plans are *universal* and *complete*. This means that plans contain paths from every state and if multiple (loop-free) paths exist from a state to the goal, then all such paths are included. Instead of immediately choosing the shortest path to the goal (which may, in practice, be sub-optimal), we shall allow the reinforcement learner to choose between them.

Plans are trees: the nodes represent sets of states; the edges represent actions. In any given state one or more nodes of the plan may be active, indicating that the corresponding action (on the outgoing edge of the node) is appropriate to execute. A node is active if it is the shallowest node on a path to the root with a true condition. Figure 2 shows two active nodes for the agent in its starting state. Both dictate the same action, Go(study, hall), but each forms part of a different path to the goal.

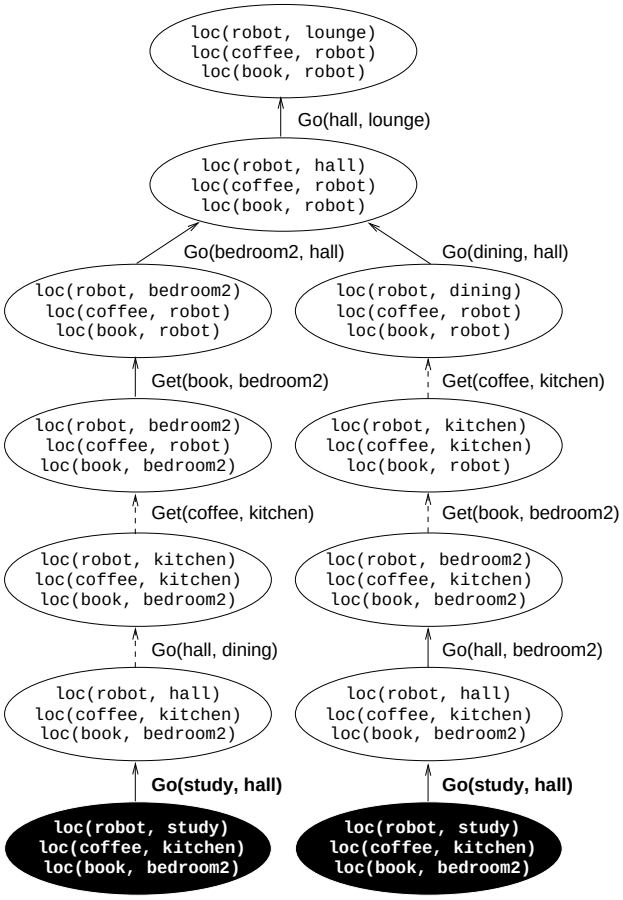

Figure 2. A partial view of the plan generated for the grid-world task. Dashed arrows indicate where nodes have been omitted. The full plan has 50 nodes and is 12 levels deep. The inverted nodes are active in the agent's starting state.

3.1 Combining Planning and Learning

To combine planning and learning we need some way to reconcile the different models of behaviours used by the two approaches. The Reinforcement Learnt Teleo-operator (or RL-Top) (Ryan & Pendrith, 1998) is such a model. It provides an abstract description for a behaviour, implemented through reinforcement learning. The same description is used for both planning with the behaviour and learning its internal policy.

As teleo-operators are only able to describe behaviours which aim to *achieve* certain goals (as opposed to behaviours which aim to maintain conditions), we shall limit our attention to these. (This is also the most commonly occurring kind of behaviour in HRL problems, so this limitation is not too strict.)

The pre-image of a teleo-operator corresponds directly to the application space of a reinforcement learnt behaviour. The post-condition of the operator denotes the local goal of the behaviour. In a recursively optimal learning algorithm this corresponds to the reward function for the behaviour. We shall assume that the object of the behaviour is to reach the post-condition as quickly as possible, without prematurely leaving the pre-image. To learn such a behaviour, we derive a reward function $\mathsf{B}.r$ for behaviour B. If the agent executes action a in state s, which results in a transition to state s' then the local reward for behaviour B is given by:

$$\mathsf{B}.r(s,a,s') = \begin{cases} 1 & \text{if } s' \models \mathsf{B}.\text{post} \\ -1 & \text{if } s' \not\models \mathsf{B}.\text{post and } s' \not\models \mathsf{B}.\text{pre} \\ 0 & \text{otherwise} \end{cases}$$

A similar equation is used to derive a global reward function from a symbolic representation of a goal G:

$$r(s,a,s') = \begin{cases} 1 & \text{if } s' \models G \\ 0 & \text{otherwise} \end{cases}$$

Table 1. Planned HSMQ-Learning

function P-HSMQ(goal G)
  plan $P \leftarrow$ BUILDPLAN(G)
  $t \leftarrow 0$
  Observe state s_t
  $\mathcal{B}_t \leftarrow$ ACTIVEBEHAVIOURS(P, s_t)
  while $s_t \not\models G$ **do**
    $T \leftarrow t$
    Choose behaviour $\mathsf{B} \leftarrow \pi(s_t)$ from $\mathcal{B}_t$
      according to an exploration policy
    sequence $S \leftarrow \{\}$
    while $s_t \models \mathsf{B}.\text{pre} \wedge s_t \not\models \mathsf{B}.\text{post}$ **do**
      Choose primitive action $a_t \leftarrow \mathsf{B}.\pi(s_t)$
        according to an exploration policy
      Execute action a_t
      Observe next state s_{t+1}
      $\mathsf{B}.Q(s_t, a_t) \xleftarrow{\alpha} \mathsf{B}.r(s_t, a_t, s_{t+1}) + \gamma \max_{a \in \mathsf{B}.\mathcal{A}} \mathsf{B}.Q(s_{t+1}, a)$
      $S \leftarrow S + \langle s_t, a_t, s_{t+1} \rangle$
      $t \leftarrow t + 1$
    end while
    $k \leftarrow 0$
    $totalReward \leftarrow 0$
    for each $\langle s, a, s' \rangle \in S$ **do**
      $totalReward \leftarrow totalReward + \gamma^k r(s, a, s')$
      $k \leftarrow k + 1$
    end for
    $\mathcal{B}_t \leftarrow$ ACTIVEBEHAVIOURS(P, s_t)
    $Q(s_T, \mathsf{B}) \xleftarrow{\alpha} totalReward + \gamma^k \max_{\mathsf{B}' \in \mathcal{B}_t} Q(s_{T+k}, \mathsf{B}')$
  end while
  return S
end P-HSMQ

3.2 Planned Hierarchical Semi-Markov Q-Learning

Armed with a notation for describing behaviours that is compatible with both symbolic planning and hierarchical reinforcement learning, we shall now examine the first of two algorithms which combines the two. Table 1 shows the pseudocode for Planned Hierarchical Semi-Markov Q-Learning [1] (P-HSMQ). This is an adaptation of Dietterich's HSMQ algorithm (Dietterich, 2000b) which uses the plan in the place of a task hierarchy. For the sake of simplicity, the algorithm has been limited to a single intermediate level of behaviours. Expanding this algorithm (and the later one, TRQ) to several levels of hierarchy is possible, but there is insufficient space in this paper to describe this in detail.

For those unfamiliar with HSMQ, it is a simple expansion of the Semi-Markov Decision Problem Q-Learning (SMDPQ) algorithm of Sutton et al. (1999). Whereas SMDPQ uses hard-coded behaviours, HSMQ learns them recursively, each according to its own local reward function. Like Dietterich's more widely known MAXQ algorithm, it is guaranteed to converge to the recursively optimal policy, given the standard assumptions about learning rates, exploration and the Markov properties of the world. We have selected this algorithm as a baseline for this work due to its relative simplicity (compared, for example, to MAXQ).

P-HSMQ contains a very simple modification to HSMQ. Where HSMQ would have consulted the task hierarchy to determine which behaviours it had to choose from, P-HSMQ builds a plan and consults it. The ACTIVEBEHAVIOURS function returns the set of behaviours dictated by the active nodes of the plan (with duplicates removed). One of these behaviours is then selected by the reinforcement learning algorithm for execution. A behaviour B learns its policy as it executes, using its local reward function, until it terminates, either successfully (satisfying B.post) or unsuccessfully (prematurely leaving B.pre). The experiences gathered while executing the behaviour and evaluated using the global reward function and used to update its global Q-value.

4. Experiment 1

To demonstrate the value of using the plan in this way, we have run experiments comparing three different approaches to the grid-world problem above:

1. HSMQ-learning with no pruning (all applicable behaviours are available)
2. Executing the plan directly with reinforcement learning only at the bottom of the hierarchy (learning primitive policies for behaviours). Choices between behaviours in the plan were resolved in favour of the shallower node, breaking ties randomly.
3. P-HSMQ-learning with a plan-based task-hierarchy

Each approach was run twenty times, with each run consisting of 1000 consecutive trials. A trial begins with the agent empty-handed at its starting location in the study, and ends when the agent managed to arrive in the lounge with both the coffee and the book. A small error was added to the robot's movement which meant that each time it moved there was a 1 in 20 chance that it would end up moving in a direction perpendicular to its intention.

[1] In the pseudocode of this paper I am using the notation $X \xleftarrow{\alpha} Y$ as a shorthand for the exponentially weighted rolling average update operation $X \leftarrow (1-\alpha)X + \alpha Y$. I believe this notation both simplifies and clarifies the equations.

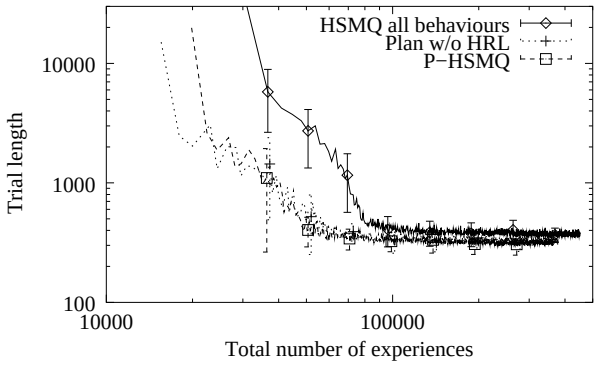

Figure 3. A comparison of learning rates for three approaches to the grid-world problem: (1) Using HSMQ to select from every applicable behaviour, (2) Using a plan to select behaviours, (3) Using P-HSMQ to select behaviours from alternatives provided by a plan. Error bars represent one standard deviation.

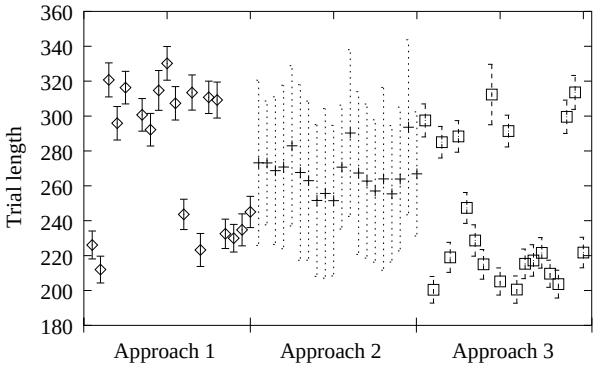

Figure 4. Average trial lengths for each individual run of the three approaches in experiment 1. Error bars indicate one standard deviation.

The learning parameters were set as follows: The learning rate α was 0.1. The discount factor *gamma* was 0.95. Exploration was done in an ϵ-greedy fashion with a 1 in 10 chance of the agent choosing an exploratory action (both at the level of primitive actions, and in the choice of behaviours). Exploratory actions were chosen using recency-based exploration.

Figure 3 shows the results of this experiment (note: this graph is plotted on a log-log scale, to highlight the differences in the early trials). Clearly the plan-based approaches converge much more rapidly than the unplanned approach. Measuring the average number of experiences required before trial-length falls below 500 shows that approach 1 takes 72324 primitive actions, approach 2 which takes 29956, and approach 3 which takes 35151. In both cases the difference is significant with 99% confidence. The reason for this difference is apparent. Approach 1 spent much of its time in early trials learning behaviours never featured in its final policy. Approaches 2 and 3 restricted their exploration to a smaller set of behaviours, which were more relevant to the task at hand. The long term performance of approach 1 is also poorer, as it continues to explore behaviours which do not contribute to the goal.

The exploratory actions performed by approaches 1 and 3 hide differences between the final policies learnt by each. To resolve this, the learnt policies from each of the three approaches were run for a further 1000 trials without any further learning or exploration. The results of these trials are shown in Figure 4.

This graph shows the average trial lengths for each repeat run. Notice that the results from both HSMQ approaches fall into two clusters, one below 250 and one above 280. These correspond to the two high-level solutions to the problem: either getting the coffee first then the book (the shorter solution) or else getting the book then the coffee (the longer solution). Both approaches converged to a policy which implemented one of these two solutions well (indicated by the small deviation per run). Approach 1, without the plan, chose the shorter solution in 8 out of 20 runs. Approach 3, with the plan, chose the shorter solution in 13 out of 20 runs. Contrast this with approach 2, which does no learning at the behaviour level. It does not settle on one solution or the other, but selects one randomly for each trial. This is shown by the much greater standard deviation in these runs.

Ideally both approaches 1 and 3 ought always to converge to the better of the two solutions. The failure to do so is probably due to lack of exploration, and the "lock-in" effect that occurs when Q-values are pessimisticly initialised. This is expected to be more of a problem with the unplanned approach as it has more options to explore and thus will take longer to find the better solution.

Even so, the combination of planning and hierarchical learning shown in approach 3 appears on average, to converge to a better and more stable solution than either planning or hierarchical learning alone.

5. Termination Improvement

While the P-HSMQ algorithm is effective, it does not make full use of the information available to it in the plans it builds. It only checks the appropriateness of a behaviour when it is initiated, and always executes it until completion, ignoring any effects that might cause the action to no longer be appropriate.

Consider the following scenario: we repeat the grid-

world experiment above but add a further complexity. There is a bump on the floor near the west end of the hall, indicated by the shaded area. When the robot moves onto the bump while carrying a cup of coffee, there is a 10% chance that it spills.

Suppose the agent has already visited the kitchen and collected the coffee. It re-enters the hall and starts executing Go(hall, bedroom2), as dictated by the plan, but as it passes over the bump it spills the coffee. Once the P-HSMQ algorithm has begun executing a behaviour it is committed to completing it, so the robot continues down the hall to the bedroom. Only once it enters the room does it re-evaluate its choice of behaviour, and realises that it needs to return to the kitchen to fetch another cup.

A more efficient solution would be to return to the kitchen as soon as the coffee is spilt. To learn to do so, the agent would need to relax its commitment to behaviours, and be able to interrupt a behaviour before it terminates. This is called *termination improvement*, and evidence shows that it can lead to better policies (Sutton et al., 1999). However the more often the agent is able to reselect its behaviour, the longer the learning process takes. Optimally we would like the agent to only reconsider its choice when the current behaviour is no longer worthwhile.

The plan provides us with a way to make this decision. Each node dictates the conditions which make its corresponding behaviour appropriate. So long as the node is active, the behaviour should continue executing. Should the node become inactive, then the behaviour may no longer be appropriate and should be interrupted. Nilsson (1994) calls this kind of execution semantics *teleo-reactive*. What follows is an HRL algorithm which implements these semantics, called Teleo-Reactive Q-Learning.

6. Teleo-Reactive Q-Learning

Teleo-Reactive Q-Learning (TRQ), as outlined in Table 2, is an adaptation of P-HSMQ which implements the teleo-reactive execution semantics described above.

There are two important issues that need to be dealt with in implementing this algorithm. They are: (1) maintaining the Semi-Markov Property of interrupted behaviours (necessary for the correctness of the SMDPQ update rule); and (2) ensuring that behaviour's internal policies are fully explored in spite of interruptions. Each of these issues is detailed below.

Table 2. TRQ-Learning

```
function TRQ(goal G)
    plan P ← BuildPlan(G)
    t ← 0
    Observe state s_t
    N_t ← ActiveNodes(P, s_t)
    while s_t ⊭ G do
        T ← t
        Choose node N ← π(s_t) from N_t
                according to an exploration policy
        sequence S ← {}
        B ← N.behaviour
        while N ∈ N_t do
            Choose primitive action a_t ← B.π(s_t)
                    according to an exploration policy
            Execute action a_t
            Observe next state s_{t+1}
            B.Q(s_t, a_t) ←α B.r(s_t, a_t, s_{t+1})
                                +γ max_{a∈B.A} B.Q(s_{t+1}, a)
            S ← S + ⟨s_t, a_t, s_{t+1}⟩
            t ← t + 1
            N_t ← ActiveNodes(P, s_t)
        end while
        k ← 0
        totalReward ← 0
        for each ⟨s, a, s'⟩ ∈ S do
            totalReward ← totalReward + γ^k r(s, a, s')
            k ← k + 1
        end for
        Q(s_T, N) ←α totalReward
                        +γ^k max_{N'∈N_t} Q(s_{T+k}, N')
        if s_t ⊨ B.pre ∧ s_t ⊭ B.post then
            with probability η do
                Persist(B)
            end with
        end if
    end while
end TRQ
```

6.1 Maintaining the Semi-Markov property

The correctness of the SMDPQ-Learning update rule used in TRQ requires that the behaviours executed obey the Semi-Markov property, that is that the outcomes of executing a behaviour – its duration, the rewards it receives and the state it terminates in – depend only on the identity of the behaviour and the state in which it is initiated. The teleo-reactive execution semantics violate this condition. If there are two different nodes active in the same state, dictating the same behaviour but with different interruption criteria, then the outcome of the executing the behaviour will depend on which node is chosen.

Sutton et al. (1999) show how to handle this situation. New "options" need to be added to represent the different possible interruption conditions placed on a behaviour. These options will share an underlying primitive policy, but need to be treated separately when learning which to select. Enough information needs to

be contained in each option description to ensure that it obeys the Semi-Markov property.

In the case of teleo-reactive execution we can use the individual nodes of the plan as options. A particular node of the plan always executes the same behaviour and terminates under the same conditions, so it satisfies the Semi-Markov property. Thus Teleo-reactive Q-Learning assigns Q-values to nodes of the plan, rather than to behaviours. Execution then consists of finding all active nodes in the plan and selecting the one with the best Q-value. The behaviour corresponding to this policy node is then executed and learnt in the usual way. When the node is no longer active the behaviour is interrupted, whether is has terminated or not, and the gathered experience is used to update the value of the node.

Table 3. TRQ-Learning: Persisting with a behaviour

function PERSIST(behaviour B)
  while $s_t \models \mathsf{B.pre} \wedge s_t \not\models \mathsf{B.post}$ **do**
    Choose primitive action $a_t \leftarrow \mathsf{B}.\pi(s_t)$
      according to an exploration policy
    Execute action a_t
    Observe next state s_{t+1}
    $\mathsf{B}.Q(s_t, a_t) \xleftarrow{\alpha} \mathsf{B}.r(s_t, a_t, s_{t+1})$
      $+\gamma \max_{a \in \mathsf{B}.\mathcal{A}} \mathsf{B}.Q(s_{t+1}, a)$
    $t \leftarrow t + 1$
  end while
end PERSIST

6.2 Persisting with interrupted behaviours

In a recursively optimal HRL algorithm such as TRQ the policy of a behaviour is independent of its calling context. A behaviour aims to learn a policy to maximise its local rewards, regardless of its parent task. To ensure that this happens, behaviours must be allowed to fully explore their application spaces. In particular, as the reward functions we have defined for behaviours only provide non-zero rewards when the behaviours terminate, behaviours must be allowed to execute to completion infinitely often in the limit, in order to guarantee convergence.

The teleo-reactive semantics alone will not guarantee this, so TRQ occasionally elects to explore a behaviour to completion rather than interrupt it. If a behaviour is interrupted before it terminates, then the algorithm may decide, with probability η to ignore the interruption and persist with the behaviour until it succeeds or fails.

Once the algorithm begins to persist with a behaviour, all of the hierarchy above that behaviour is ignored. The value of the node that was dictating the behaviour is updated as if the behaviour was interrupted. Experiences gathered while persisting with a behaviour are only used to update that behaviours internal Q-values, and are ignored by the level above. In this way, persistence is a kind of off-policy exploration, at the behaviour level.

When behaviour has terminated, either successfully or unsuccessfully, control returns to the plan and a new plan node is selected according to the next state.

7. Experiment 2

Our second experiment compares P-HSMQ and TRQ. The experimental set-up is the same as described previously, except that whenever that the robot steps into one of the shaded squares marked "Bump" it has a 10% chance of spilling the coffee if it is carrying it. Once the coffee is spilled it returns to the kitchen.

Two approaches were compared: one using P-HSMQ and one using TRQ. Learning and exploration rates were set as in Experiment 1. In the TRQ approach, the value of η was set at 0.1 [2]. Twenty independent runs are performed for each approach. Each run consists of 1000 learning trials, and then 1000 test trials (in which learning is disabled and both ϵ and η are set to zero).

Convergence times for both approaches are comparable. P-HSMQ took on average 55911 experiences to produce to a trial length less than 500 steps, TRQ took 45657 experiences. The difference between these results is not significant.

Examining the results of the replayed trials, however, does produce a significant result. Dividing the trials into those in which the coffee was spilled and those in which it was not, and averaging trial lengths in each group gives us the results shown in Table 4.

Table 4. Average trial lengths and standard deviations for final learnt policies from Experiment 2.

	P-HSMQ	TRQ
Spill	472.25 ± 33.68	347.01 ± 38.26
No Spill	296.61 ± 23.54	283.44 ± 37.67
Difference	175.64 ± 23.96	63.57 ± 22.82

When there is no spill the results are similar, as we would expect them to be, but the P-HSMQ shows a significantly greater average trial length over TRQ in those cases where a spill occurred (with 99% confidence). Committing to behaviours has on average

[2] Other values for η, ranging from 0 to 0.5 were tested but had no significant effect on the results. This is somewhat surprising, but time does not permit a full analysis to be included here.

made P-HSMQ take 112 steps more to recover from a spill than TRQ. This demonstrates how termination improvement, applied appropriately according to the plan, can result in more efficient policies with little extra learning time.

8. Related Work

Model-based reinforcement learning has a long history and other work has been done applying this to hierarchical policies, (eg. Sutton et al., 1999; Mahadevan et al., 1997; Singh, 1992). Such algorithms begin with operational behaviours and then learn (or otherwise acquire) concrete, numeric models of these behaviours from which behaviour-based policies can be constructed. The work in the paper turns this idea on its head, beginning with abstract models of desired behaviours and learning behaviours to satisfy these models. As our models lack concrete details, they are only used to guide exploration of behaviour-based policies, and the concrete details are learnt by reinforcement learning.

This work bears strong similarity to (Boutilier et al., 1997) in that it aims to bring together classical and decision theoretic planning techniques. However unlike that work, this paper treats behaviours as temporally abstract and having an underlying implementation in terms of primitive actions. We have only recently become aware of this relationship, and we imagine much fruit may come from a detailed investigation thereof.

9. Conclusions and Future Work

We build learning agents ultimately so that they can learn things that would be too hard for us to program. Yet we want them to take advantage of what knowledge we do have and can easily communicate. Humans communicate best in abstract form, so building tools which can take abstract advice and turn it into concrete knowledge is an important part of AI.

In this paper we have presented an approach to such a system. The individual elements of the system are not new. We have merely shown how existing tools in planning and hierarchical reinforcement learning can be made to operate side-by-side and complement each other. Planning allows us to automatically construct task hierarchies for reinforcement learning, and to select intelligently from a large and varied repertoire of behaviours. Hierarchical reinforcement learning allows us to automatically learn concrete policies for abstractly described behaviours and to choose intelligently between different paths in the plan. The resulting hybrid has been shown to learn faster and produce better policies than either approach alone.

There are many more ways in which having both an abstract and a concrete model of behaviours can add to an agent's ability to intelligently interact with its world. A high-level plan gives the agent a sense of why it is executing a particular behaviour and the ability to diagnose what went wrong when the plan fails (see (Reid & Ryan, 2000) for an example of this). Further reasoning of this kind might allow agents to invent new behaviours of their own, to fit the circumstances that arise.

References

Boutilier, C., Brafman, R. I., & Geib, C. (1997). Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence*. San Franciso, CA: Morgan Kaufmann.

Dietterich, T. G. (2000a). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Artificial Intelligence*, *13*, 227–303.

Dietterich, T. G. (2000b). An overview of MAXQ hierarchical reinforcement learning. *Proc. of the Symposium on Abstraction, Reformulation and Approximation* (pp. 26–44). New York, NY: Springer Verlag.

Mahadevan, S., Khaleeli, N., & Marchalleck, N. (1997). Designing agent controllers using discrete-event markov models. *AAAI Fall Symp. on Model-Directed Autonomous Systems*. Cambridge, MA.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, New Jersey: Prentice-Hall.

Nilsson, N. J. (1994). Teleo-reactive 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*. The MIT Press.

Reid, M. D., & Ryan, M. R. K. (2000). Using ILP to improve planning in hierarchical reinforcement learning. *Proc. 10th Int. Conf. on Inductive Logic Programming* (pp. 174–190). Springer-Verlag.

Ryan, M. R. K., & Pendrith, M. D. (1998). RL-TOPs: an architecture for modularity and re-use in reinforcement learning. *Proc. 15th Int. Conf. on Machine Learning* (pp. 481–487). Morgan Kaufmann, San Francisco, CA.

Singh, S. P. (1992). Reinforcement learning with a hierarchy of abstract models. *Proc. of the 10th Nat. Conf. on Artifical Intelligence*. Cambridge, MA: MIT Press.

Sutton, R. S., Precup, D., & Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.