Defining Object Types and Options Using MDP Homomorphisms

Alicia Peregrin Wolfe PIPPIN@CS.UMASS.EDU
Andrew G. Barto BARTO@CS.UMASS.EDU
Computer Science Department, University of Massachusetts, Amherst, MA 01003 USA

Abstract

Agents in complex environments can have a wide range of tasks to perform over time. However, often there are sets of tasks that involve similar goals on similar objects, e.g., the skill of making a car move to a destination is similar for all cars. This paper lays out a framework for specifying goals that are parameterized with focus objects, as well as defining object *type* in such a way that objects of the same type share policies. The method is agnostic as to the underlying state representation, as long as simple functions of the state of the object can be calculated.

1. Introduction: Modeling Objects

We typically categorize objects in our environment into a set of types, defined by their behavior: cars are objects that we can drive; cups hold liquid; chairs can be used to sit on. This kind of object type definition is particularly useful to an agent learning to function in a complex environment: the type labels provide useful abstractions over the details of each object's features.

In this work, we use the MDP homomorphism (Ravindran, 2004) framework to determine which objects have the same type. Type is defined here only relative to some task or class of tasks: for example, objects sharing the same type for tasks involving their location may not share the same type for color. From this we show how to construct options that execute over objects of a variety of types, with one policy stored for each allowable type of object.

An option defined for a particular type can be reused on new objects, as long as they fit the criteria for that object type. Once we know how to pick up a cup and drink, this skill can be applied to any cup.

Appearing in *the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

2. MDP and CMP Homomorphisms

A Markov Decision Process (MDP) is a tuple (S, A, T, R) consisting of a state set (S), action set (A), transition function ($T : S \times A \times S \to [0, 1]$), and expected reward function ($R : S \times A \to \mathbb{R}$). The transition function defines the probability of state transitions given a chosen action, while the reward function gives the expected immediate reward the agent receives for executing an action in a particular state.

An MDP homomorphism (Ravindran, 2004) is a mapping, $h : S \times A \to S' \times A'$, from the states and actions of a base MDP, $M = (S, A, T, R)$, to an abstract model MDP $M' = (S', A', T', R')$. The mapping h consists of a set of mappings: $f : S \to S'$, and for each $s \in S$ a mapping $g_s : A \to A'$ that recodes actions in a possibly state-dependent way. The following properties must hold for all state and action pairs:

$$R'(f(s), g_s(a)) = R(s, a) \tag{1}$$

$$T'(f(s_i), g_{s_i}(a), f(s_j)) = \sum_{s_k | f(s_k)=f(s_j)} T(s_i, a, s_k). \tag{2}$$

Subgoal *options* (Sutton et al., 1999) provide a formalism for specifying multiple episodic subtasks within an MDP. In addition to a reward function $\mathbb{R}$, a subgoal option includes a termination condition $\beta : S \times A \to [0, 1]$ which specifies the probability that the option will terminate in any particular state. Homomorphisms for subgoal options add a constraint for β (Ravindran & Barto, 2003). For all $s \in S$ and $a \in A$:

$$\beta'(f(s), g_s(a)) = \beta(s, a). \tag{3}$$

When a mapping f can be found that is many-to-one, the abstract MDP M' has fewer states than M. The homomorphism conditions mean that M' accurately tracks the transitions and rewards of M but at the resolution of blocks of states, assuming some appropriate action recoding. These properties guarantee that policies optimal for M' can be *lifted* to produce optimal policies of the larger MDP M (Ravindran, 2004).

A Controlled Markov Process (CMP) with output is a tuple (S, A, T, y), where S, A, and T are as in an MDP, and y is an output function $y : S \times A \rightarrow Y$ (Wolfe & Barto, 2006). We think of the output function as singling out some aspect of the CMP as being of interest. This function might be, for example, the location or color of an object in the state. Given any function $r : Y \rightarrow \mathbb{R}$, $(S, A, T, r \circ y)$ is an MDP whose reward function is the composition of r and y. We say that this MDP is *supported by* y. The termination conditions for the family of subgoal options supported by y have the form $\beta : Y \mapsto [0, 1]$.

A CMP homomorphism is a mapping from a CMP with output (S, A, T, y) to an abstract CMP with output (S', A', T', y'). The mapping functions h, f, and g_s are defined as for MDP homomorphisms. The constraints over the reward function (Equation 1) and termination function β (Equation 3) are replaced by a single constraint over the output function. For all $s \in S$ and $a \in A$ the following must hold:

$$y'(f(s), g_s(a)) \quad = \quad y(s, a). \tag{4}$$

The transition function constraints (Equation 2) do not change. The model formed by a CMP homomorphism can be used to learn a policy for any supported reward $(r \circ y)$ and termination $(\beta \circ y)$ functions.

Several algorithms exist for finding MDP homomorphisms given a model of an MDP, and can be trivially adapted to find CMP homomorphism. All proceed by partitioning the states and actions into two sets of blocks: a state (S) partition $\{B_1, ...B_m\}$ over states, and a state/action (SA) partition over (s, a) pairs, $\{P_1...P_n\}$. The S partition defines an f mapping: $s \in B_i \rightarrow f(s) = s'_i$. Similarly, the SA partition defines the set of g_s mappings: $(s, a) \in P_i \rightarrow g_s(a) = a'_i$. The version of the homomorphism finding algorithm used in this paper is taken from (Ravindran, 2004), though similar examples exist in (Givan et al., 2003) and (Boutilier et al., 2001).

3. CMPs with Objects

At this point, we have most of the basic machinery we need to model environments with objects. The main addition made in this section is a transformation of the way we encode the state space: rather than being "global", the output function will now be associated with some object in the environment. The methods we have discussed so far enable us to determine which objects have similar behavior, no matter what state description we use.

An Object CMP consists of a CMP, a set of object identifiers O, an object description set D (often factored into a set of features), and a set of functions, one per object, that maps states to object descriptions: $w_o : S \rightarrow D$. The output function z in this case maps object descriptions to outputs: $z : D \rightarrow Y$. Specifying an object yields a CMP with output, $(S, T, A, z \circ w_o)$, which can be transformed into an MDP by specifying a reward function $(r \circ z \circ w_o)$ and termination function $(\beta \circ z \circ w_o)$ as in the previous sections.

The object/state specification may be as primitive or structured as the designer wishes, as long as there is some way to compute the desired output function, given an object pointer. Take two examples: one state space made up of pixels, one of features. The mapping w_o singles out some subset of pixels or features in each state as belonging to the object o. The features that belong to a particular object are typically fixed and are often named (*object1.position*, for example), whereas the set of pixels belonging to an object might change from state to state. Nonetheless, the mapping z for object position can be calculated from this set of pixels as well as it can from a set of features.

Since there is no guarantee that all objects will be present in all states, the output function $z \circ w_o$ evaluates to a special null output function value, $\perp$, for states in which the specified object is not present. Objects can be nested and can overlap.

We assume that the w_o mappings and function z are given. A homomorphic mapping h for any CMP and object based output function $(M_k, w_o \circ z)$ can be found in the same way as it would be for any CMP with output. The interesting question is: when do multiple objects o_i and o_j share the same abstract model for z, though their state/action mapping functions h_i and h_j are different? This equivalence will be key to constructing options that operate over a class of objects.

3.1. Object Options

An object option subgoal consists of a reward and termination function: $(\beta \circ z, r \circ z)$, defined for a particular object feature z. For example, in a Blocks World CMP one option subgoal might be to move the focus block to a particular location. The reward function would map the output function $z =$ "block position" to a reward that is positive when the desired location is achieved and negative otherwise.

All other parts of the option structure follow from the reward function and termination function. Applying the subgoal to a CMP M_i and object o_j yields a subgoal option MDP $(S_i, A_i, T_i, \beta \circ z \circ w_{o_j}, r \circ z \circ w_{o_j})$, from which a policy can be calculated.

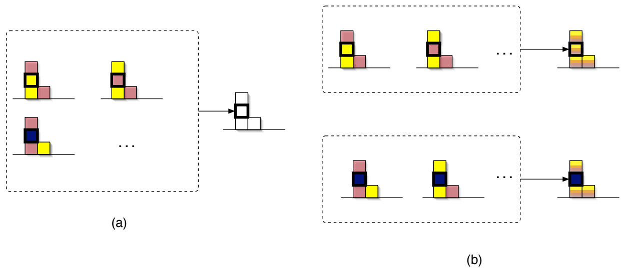

Figure 1. Example clusters from a blocks world with 4 blocks. Blocks in (a) all behave the same way, blue blocks (the dark blocks) in (b) are more slippery. Dashed squares represent blocks of the two state partitions.

We will denote by $h(M)$ the model formed by applying h to the CMP M. Consider what happens when h_i and h_j are valid homomorphisms for $z \circ w_{o_i}$ and $z \circ w_{o_j}$ respectively, and $h_i(M_k) = h_j(M_l) = M_a$. Any optimal policy in M_a for a task $(r \circ z, \beta \circ z)$ can be *lifted* to M_k using h_i^{-1} for o_i and to M_l using h_j^{-1} for o_j. The agent therefore only needs to store one policy which works for both pairs (M_k, o_i) and (M_l, o_j). If two objects have exactly the same abstract model set over all CMPs in which they could appear, we say that they have the same type for z.

Because it is also two objects share models for only a subset of the CMPs in which they could appear, our definition of object type forms a partial ordering over objects. If the function u maps objects in O onto a partially ordered set of type symbols, we would like to find a *consistent* type mapping for the output function z and set of CMPs $C = \{M_i\}$. Consistent type mappings have the following property: if $u(o_i) \preceq u(o_j)$ then for any M_k where h_i is a homomorphic mapping for $z \circ w_{o_i}$, there is a CMP M_l for which h_j is a homomorphic mapping for $z \circ w_{o_j}$ and produces a model with the following property: $h_i(M_k) = h_j(M_l)$.

Figure 1 compares partition blocks for two different blocks world state spaces: one in which all blocks have identical behavior, and one in which blue blocks are more slippery than others. One consistent u mapping for the output function "block position" for Figure 1b is u(blue blocks) $= \alpha$, u(other colors) $= \gamma$, $\alpha \neq \gamma$. In Figure 1a all blocks have the same type.

Policies learned for one object can be applied to any object of the same or lesser type. The option may specify a single type, or a range of types allowable for the focus object. The option must store a separate policy for each type of object which is allowable as a parameter. The generic policies are then mapped back to the true CMP by the same *lifting* process used in MDP homomorphisms.

Despite changes in the policy, the reward function is consistent across all possible parameter assignments. The goal of moving a block to position k is the same, whether the block is blue or green—only the method of executing that goal and the probability of success change if the type of the focus object changes.

3.2. Algorithm

To find reductions that tell us which objects have the same type, some modifications to the original homomorphism-finding algorithms are necessary.

We start with an output function z, and a set of one or more object CMP models $C = \{M_i\}$, each of which has a set of objects O_i. For each sample CMP M_i, we construct a set of $n = \|O_i\|$ CMPs of the form $(S_i, A_i, T_i, z \circ w_{o_k})$. The essential question is: which CMP/object pairs are isomorphic?

The algorithms for finding homomorphisms can be executed over multiple CMPs at once: the algorithm simply considers the combination of the two state and action spaces and proceeds. If, upon termination, the states and actions of two CMPs map to the same abstract states and actions (see Figure 1), their reduced models are isomorphic.

4. Experiments

All of these experiments used the algorithm above to create a library of models for 3-block blocks worlds. In each case the dynamics of the environment were changed to create a different type mapping.

For the 3-block blocks world with dynamics matching Figure 1b, the algorithm correctly finds two types of blocks, with 6 abstract CMP models, 3 used by blue blocks and 3 by focus blocks of other colors.

For our second example, consider a blocks world in which all blocks of any given color c stick to blocks of the same color. When a block sticks to the block beneath it, the probability of successfully lifting it and moving it to another pile is lower. While all blocks in this example have the same type, their abstract state mapping function is different: two yellow blocks with a blue focus block yields different dynamics than the same CMP with a yellow focus block. The algorithm finds the correct set of 4 abstract CMPs: M'_1 in which all three blocks have different colors (30 states), M'_2 in which the other two blocks match each other but not the focus block (30 states), M'_3 in which one other block matches the focus block color (60 states) and M'_4 in which all three blocks have the same colors (30

Table 1. Abstract CMPs for the blocks world in which blue and green blocks stick to other blocks of the same color.

	Models Used by Focus Block			
Focus:	M'_1 (30)	M'_2 (30)	M'_3 (60)	M'_4 (30)
blue	✓	✓	✓	✓
green	✓	✓	✓	✓
red	✓	✓		
yellow	✓	✓		

states).

Finally, for a simple example of types which are partially ordered, consider a blocks world in which blue and green blocks stick to blocks of the same color, while all other combinations of blocks interact normally. This results in 4 abstract CMPs, shown in Table 1, and two object types. The first type (blue or green focus blocks) uses all 4 models, while the second type (other blocks) uses only the two simpler models.

As expected, learning policies for supported reward functions in the reduced models is faster and has the same optimality guarantees as learning in the complete model, however, due to lack of space we do not include the results here.

5. Discussion

One alternative to the method above would be to simply add the object pointer to the state space and use the resulting state space for the option. This would result in one large reduced CMP, consisting of the union of the CMPs for each type of object. Planning in this large CMP would be significantly slower — it is much more efficient to divide the CMP into its component pieces for each object type (none of which interact) and solve each individually.

Another advantage to modeling type explicitly is that the features that determine the type of an object do not change over time, if type is defined as it is defined here. This means that these features can be examined once at the outset of the option using an object, but do not need to be reexamined after the MDP corresponding to the type has been chosen. A similar argument could be made for the objects which interact with the focus object — they also have "type" though it is not defined in the same way as for focus objects. One of the next elements of this research will be to define type for related objects, as well.

We focused in this paper on the type of focus objects because we wished to define the minimum amount of communication necessary between the upper level option and lower level. Object parameters define a limited, but useful, range of control for the upper level option, while leaving to the option the definition of all related state information. The upper level option controls the reward function of the lower level option through the option parameters, but not the policy or state space, though these are completely determined by the reward. This translation from object pointer/option id to reward function and state space/policy enables the upper level option to select objects based on criteria which can be completely orthogonal to the features used within the option.

In order to use these parameterized options in the context of a higher level task, the agent must learn to assign objects to the pointers. This is closely tied to the notion of *deixis* (Agre & Chapman, 1987).

If there are some features of the object which cannot be observed which determine its true type (if the color of blue blocks in Figure 1 was unobserved) the agent is in a special type of POMDP in which the type of the object must be discovered through interaction.

6. Acknowledgements

This research was facilitated in part by a National Physical Science Consortium Fellowship and by stipend support from Sandia National Laboratories, CA. This research was also funded in part by NSF grant CCF 0432143.

References

Agre, P. E., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. *Proceedings of the 6th National Conference on Artificial Intelligence.*

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order mdps. *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (pp. 690–697).

Givan, R., Dean, T., & Greig, M. (2003). Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, *147*, 163–223.

Kersting, K., & Raedt, L. D. (2003). Logical markov decision programs. *Proceedings of the 20th International Conference on Machine Learning.*

Ravindran, B. (2004). *An algebraic approach to abstraction in reinforcement learning.* Doctoral dissertation, University of Massachusetts.

Ravindran, B., & Barto, A. G. (2003). Smdp homomorphisms: An algebraic approach to abstraction in semi markov decision processes. *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (pp. 1011–1016). AAAI 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.

Wolfe, A. P., & Barto, A. G. (2006). Decision tree methods for finding reuseable mdp homomorphisms. *Proceedings of the 21st National Conference on Artificial Intelligence.* To appear.