Carnegie Mellon University
Research Showcase @ CMU

Institute for Software Research | School of Computer Science

2009

Skill Reuse in Lifelong Developmental Learning

Jonathan Mugan
Carnegie Mellon University, jmugan@andrew.cmu.edu

Benjamin Kuipers

Follow this and additional works at: http://repository.cmu.edu/isr

Published In
.

This Working Paper is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Institute for Software Research by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Skill Reuse in Lifelong Developmental Learning

Jonathan Mugan
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
jmugan@cs.utexas.edu

Benjamin Kuipers
Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109 USA
kuipers@umich.edu

***Abstract*— Development requires learning skills using previously learned skills as building blocks. For maximum flexibility, the developing agent should be able to learn these skills without being provided an explicit task or subtasks. We have developed a method that allows an agent to simultaneously learn hierarchical actions and important distinctions autonomously without being specified a task. This ability to learn new distinctions allows it to work in continuous environments, and allows an agent to learn its first actions from motor primitives.**

In this paper we demonstrate that our method can use experience from one set of variables to more quickly learn a task that requires additional new variables. It does this by learning actions in a developmental progression. In addition, we demonstrate this developmental progression by showing that actions that are mainly used for exploration when first learned are later used as subactions for other actions.

I. INTRODUCTION

Learning reusable skills is an important part of autonomous mental development [1]. For the skills to be most useful for development, they should form a hierarchy so that skills learned earlier can be used later as building blocks for more complex skills. Additionally, from the perspective of autonomous mental development, it is important that this hierarchy of skills be learned without the agent being given a specified task. This is because we desire that the agent develop a broad competency and be able to handle unforeseen circumstances. Autonomous skill learning also frees the designer from having to predict ahead of time all of the things that the robot might need to do and the subskills it will need to perform the task. The robot should therefore learn a set of skills (or affordances [2]) that are presented to it by the environment.

Bonarini *et al*. [3] and Vigorito and Barto [4] present methods for autonomously learning skills in discrete environments. In [5] we presented a method for learning hierarchical skills in a continuous environment. We believe that working in a continuous environment is important because it requires the agent to learn both its first primitive actions and a state abstraction appropriate for the environment.

The contribution of this paper is a demonstration of skill reuse in a continuous environment where the agent is not specified a task and learns actions autonomously. In this paper we evaluate how the method presented in [5] allows the agent to apply (transfer) skills learned early in development to a task later in development. We demonstrate this transfer learning using the Qualitative Learner of Perception and Abstraction, QLAP. QLAP is a constructivist algorithm that simultaneously learns a qualitative state representation and a model. The model takes the form of a set of dynamic Bayesian networks (DBNs) [6]. When a DBN sufficiently predicts the dynamics of the environment, it is used to create a qualitative action. The goal of each qualitative action $qa(v, q)$ is to set a variable v to a desired value q [5], [7].

We evaluate our algorithm using a simulation with realistic physics. The simulation consists of a robot sitting at a table with a block. We compare two agents and evaluate how well each agent can learn to hit the block off the table. The first agent initially has access to all of the variables and learns to hit the block off the table as part of its developmental progression. The second agent begins with experience with a subset of the variables and then performs developmental learning with access to all of the variables. We show that the second agent learns the task faster by using its previous experience with the limited number of variables. This is a small but important step to lifelong learning.

II. THE QUALITATIVE LEARNER OF ACTION AND PERCEPTION, QLAP

QLAP assumes that the agent can individuate and track objects and measure the distance between objects. QLAP also assumes that the agent has motor variables that can be set to cause movement. The result of this assumption is that QLAP interacts with the world using a set of continuous variables (a factored state representation). An overview of QLAP is shown in Fig. 1.

A. Qualitative Representation

QLAP learns a qualitative state abstraction appropriate to its environment [7]. A qualitative representation allows the agent to focus on important distinctions while ignoring others [8]. QLAP converts continuous variables to qualitative variables using *landmarks*. A *landmark* is a symbolic name for a point on a number line. Using landmarks we can convert a continuous variable $\tilde{v}$ with an infinite number of values into a qualitative variable v with a finite set of qualitative values $\mathcal{Q}(v)$ called a *quantity space* [8]. A quantity space $\mathcal{Q}(v) = L(v) \cup I(v)$, where $L(v) = \{v_1^*, \cdots, v_n^*\}$ is a totally ordered set of landmark values, and $I(v) = \{(-\infty, v_1^*), (v_1^*, v_2^*), \cdots, (v_n^*, +\infty)\}$ is the set of mutually disjoint open intervals that $L(v)$ defines in the real number line. A quantity space with two landmarks might be

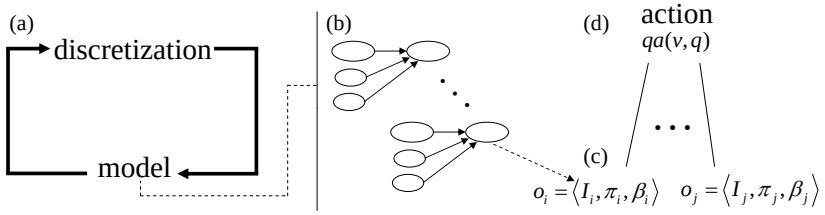

Fig. 1. (a) QLAP simultaneously learns a model and a discretization of the environment. (b) The model takes the form of many small dynamic Bayesian networks (DBNs). (c) When a DBN makes a sufficiently deterministic prediction, it is converted into a plan in the form of an option. (d) A plan (option) serves as one way to perform an action.

described by (v_1^*, v_2^*), which implies five distinct qualitative values, $\mathcal{Q}(v) = \{(-\infty, v_1^*), v_1^*, (v_1^*, v_2^*), v_2^*, (v_2^*, +\infty)\}$.

QLAP receives a set of continuous and nominal (discrete) input variables from the world and uses a set of continuous motor variables as output. Two qualitative variables are created for each continuous input variable $\tilde{v}$, a discrete variable $v(t)$ that represents the magnitude of $\tilde{v}(t)$, and a discrete variable $\dot{v}(t)$ that represents the direction of change of $\tilde{v}(t)$. Also, a qualitative variable $v(t)$ is created for each continuous motor variable $\tilde{v}$. For ease of discussion, we will consider nominal variables to be qualitative variables, and we will consider the discrete values of nominal variables to be qualitative values. The result of these transformations is four types of qualitative variables that the agent can use to affect and reason about the world: *motor variables*, *magnitude variables*, *direction of change variables*, and *nominal variables*. The properties of these variables are shown in Table I.

TABLE I
TYPES OF QUALITATIVE VARIABLES

Type of Variable	Initial Landmarks	Learn Landmarks?
motor	{0}	yes
magnitude	{}	yes
direction of change	{0}	no
nominal	not applicable	not applicable

Each direction of change variable $\dot{v}$ has a single intrinsic landmark at 0, so its quantity space is $\mathcal{Q}(\dot{v}) = \{(-\infty, 0), 0, (0, +\infty)\}$. Motor variables are also given an initial landmark at 0. Magnitude variables initially have no landmarks because zero is just another point on the number line. Initially, when the agent knows of no meaningful qualitative distinctions among values for $\tilde{v}(t)$, we describe the quantity space as the empty list of landmarks, $\{\}$. However, the agent can learn new landmarks for magnitude and motor variables. Each additional landmark allows the agent to perceive or affect the world at a finer level of granularity.

B. Events

If a is a qualitative value of a qualitative variable A, meaning $a \in \mathcal{Q}(A)$, then the *event* $A_t \rightarrow a$ is defined by $A(t-1) \neq a$ and $A(t) = a$. That is, an event takes place when a discrete variable A changes to value a at time t, from some other value. We will often drop the t and describe this simply as $A{\rightarrow}a$. We will also refer to an event as E when the variable and qualitative value involved are not important, and we use the notation $E(t)$ to indicate that event E occurs at time t.

C. Dynamic Bayesian Networks

QLAP learns DBNs to represent the dynamics of the environment [5]. The notation we use for these DBNs is $r = \langle \mathcal{C} : E_1 \Rightarrow E_2 \rangle$ where $\mathcal{C}$ is a set of qualitative variables that serves as a context, event $E_1 = X{\rightarrow}x$ is the *antecedent event*, and event $E_2 = Y{\rightarrow}y$ is the *consequent event*. This DBN r can be written as

$$r = \langle \mathcal{C} : X{\rightarrow}x \Rightarrow Y{\rightarrow}y \rangle \tag{1}$$

DBN r gives the probability that event $Y \rightarrow y$ will soon follow event $X{\rightarrow}x$ for each qualitative value in $\mathcal{C}$. Focusing only on timesteps in which event $X \rightarrow x$ occurs helps to focus the agent's attention to make learning more tractable. And using a time window for event $Y \rightarrow y$ allows the DBN to account for influences that may take more than one timestep to manifest. Notice that these DBNs differ from the typical DBN formulation, e.g. [9], in that there is no action associated with the DBN. This is because QLAP does not begin with a set of primitive actions, it only assumes that the agent has motor variables. The DBNs in QLAP are tied to the agent's motors because the antecedent event of the DBN can be on a motor variable.

To learn each DBN, QLAP finds a pair of events E_1 and E_2, such that E_2 is more likely to occur soon given that E_1 has occurred than otherwise. It then creates a DBN with an empty context. As the agent explores, QLAP then iteratively adds context variables that improve the predictive ability of the DBN (cf. marginal attribution [10]). When there is a context for which the CPT of the DBN states that event E_2 will soon follow event E_1 with a probability that exceeds 0.75, the DBN is labeled *sufficiently deterministic*.

Predicting when the consequent event will follow the antecedent event is a supervised learning problem. This formulation allows the agent to learn new landmarks (distinctions) that improve the predictive ability of the DBN (see [7], [11] for details.)

D. Actions

In QLAP, the agent learns *qualitative actions* to achieve the qualitative values of variables [5]. Each qualitative action ("action" for short) sets the qualitative value of a variable to a desired value. In QLAP, actions may be achieved in more than one way. Each way to achieve the action is called a plan. Each plan is represented as an option [12]. Once a DBN is sufficiently deterministic, it is converted into a plan. These plans are learned using reinforcement learning [13], see [5] for details. (Although, to reduce memory use, in this paper the state space of the option consists only of the context variables and does not include the antecedent and consequence variables, as was done in [11].)

III. DEVELOPMENTAL LEARNING IN QLAP

QLAP is not given a learning objective; instead, QLAP learns in a developmental progression. This developmental progression comes from incrementally learning DBNs, actions, and landmarks. As described in [14], the developmental progression also comes from restrictions that take three forms

1) *Restrictions on Learning DBNs* In QLAP, a DBN can only be learned if its antecedent event can be reliably predicted by a previously learned DBN.
2) *Restrictions on Learning Plans* A DBN can only be converted to a plan if the antecedent event can be reliably achieved using an existing action.
3) *Restrictions on Cognitive Load* An agent has limited cognitive resources and an important part of development is freeing up resources. QLAP designates an action as *open*, *full* or *closed*. An action is closed if its goal can be achieved at least 75% of the time; otherwise, it is full if it has 5 plans; and it is open otherwise. Actions that are closed or full do not accept additional plans. When an action is closed it also affects the learning of DBNs. QLAP does not add a DBN if the action to bring about the consequent event is closed. If an action is closed, all DBNs that predict that event that are not converted to plans for that action are deleted. A DBN is also deleted if it does not become a plan after 100,000 timesteps. Additionally, a plan (and its associated DBN) is deleted if the action is not closed and its reliability in no situation is above (or equal to) 5%.

A. Choices Made During Exploration

The agent continually makes three types of choices during its exploration. These choices vary in time scale from coarse to fine.

1) The agent chooses which of its learned actions to practice. This can be done randomly or by using a version of Intelligent Adaptive Curiosity (IAC) [15] which first measures the change in the agent's ability to perform the action over time and then chooses actions where that ability is increasing. In this paper, these actions are chosen randomly.
2) The agent chooses the best plan for performing the action. This is done by probabilistically choosing a plan where the probability of choosing each plan is based on its estimated probability of success.
3) The agent chooses the subaction within the plan. This is done using the standard reinforcement learning technique ϵ-greedy that balances exploration with exploitation [13].

B. Execution

An outline of the execution of QLAP is shown in Algorithm 1. Note that for the first 20,000 timesteps the agent chooses random motor babbling exploration actions. After that point it chooses a motor babbling action with probability 0.1, otherwise it chooses an exploration action randomly and chooses an action plan according to [5].

Algorithm 1 The Qualitative Learning of Action and Perception (QLAP)

```
for t = 1 : ∞ do
    Sense environment
    Convert input to qualitative values using current landmarks
    Update statistics to learn new DBNs
    Update statistics for each DBN
    if mod(t, 2000) == 0 then
        Learn new DBNs
        Delete unneeded DBNs and plans (options)
        Learn new landmarks to change qualitative representation
        Learn new actions
    end if
    if current exploration action is completed then
        Choose new exploration action and action plan
    end if
    Get low-level motor command based on plan of current exploration action
    Pass motor command to robot
end for
```

IV. EVALUATION

A. Experimental Methodology

We run experiments using the environment shown in Fig. 2. The environment is implemented in Breve [16] and has realistic physics. The simulation consists of a robot at a table with a block. The robot has an orthogonal arm that can move in the x and y directions. The agent explores autonomously. Each time the agent knocks the block out of reach the block is replaced with a different block and put on the table. The block size varies randomly in length from 1.0 to 3.0 units.

B. Evaluation Task

The task is to hit the block to the floor. QLAP learns autonomously to build task-general skills. During learning, QLAP does not know that it will be evaluated on this task.

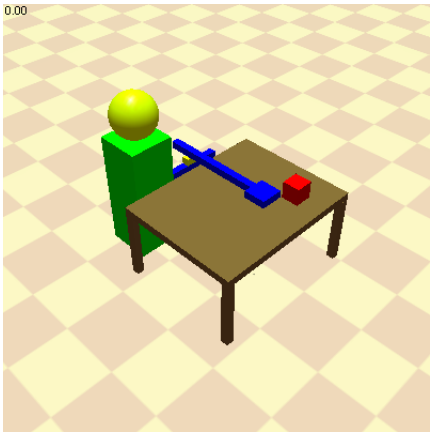

Fig. 2. The robot is implemented in Breve; a simulator with realistic physics. The robot has a torso with a 2-dof orthogonal arm and is sitting in front of a table with a block. The robot has two motor variables $\tilde{u}_x$ and $\tilde{u}_y$ that move the hand in the x and y directions, respectively. The location of the hand is given by two time-varying continuous variables $\tilde{h}_x$ and $\tilde{h}_y$ that represent the location of the hand in the x and y directions, respectively. The relationship between the hand and the block is represented by the continuous variables $\tilde{x}_{rl}$, $\tilde{x}_{lr}$, and $\tilde{y}_{tb}$. The variable $\tilde{x}_{rl}$ is the x value of the location of the right side of the hand in a coordinate system whose origin is centered on the left side of the block (variable $\tilde{x}_{lr}$ is analogous). The variable $\tilde{y}_{tb}$ is the y value of the location of the far (top) side of the hand in a coordinate system whose origin is centered on the bottom (near) side of the block. The variable $\tilde{z}_f$ represents the distance to the floor. There is also a Boolean variable $bang$ that is true when the block touches the floor ($\tilde{z}_f < 0.05$). There are also three variables $\tilde{e}_l$, $\tilde{e}_r$, and $\tilde{e}_t$ that give the closest part of the block and the left, right, and top edge of the table, respectively. Including the direction of change variables, there are 21 variables total.

We can be confident that it will learn the specified task because the environment is small. In larger environments, more care would have to be taken to ensure that the agent saw the evaluation task as sufficiently "interesting" to learn.

Every 10,000 timesteps (about every 8 minutes of physical experience) we saved the state of the agent. We then tested how well it could do that task starting from this stored learned state. Each evaluation consisted of 100 episodes. Each episode lasted for 300 timesteps or until the block was moved. The agent received a penalty of -0.01 for each timestep, and it received a reward of 10.0 if it completed the task.

C. Experimental Conditions

There are two experimental conditions. For each experimental condition we trained 20 agents.

1) **non-transfer** The agent explores for 150,000 timesteps and has access to all of the variables.
2) **transfer** The agent learns for 150,000 timesteps without the variables relevant to the task ($bang$ and z_f in Fig. 2). It then learns for another 150,000 timesteps using all of the variables.

We call the conditions **transfer** and **non-transfer** because the **transfer** agent should use what it learned during the first 150,000 to more quickly learn to perform the evaluation task compared with the **non-transfer** agent that must begin learning from scratch.

V. RESULTS

We see in Fig. 3 that the **transfer** agent can reuse the skills learned during exploration with the limited set of variables to learn to perform the task faster.

We see in Fig. 4 that the **transfer** agent makes more exploratory calls to $bang$ because it developmentally ready to knock the block off the table to hear the bang noise. By contrast, the **non-transfer** agent must reach that level of development. We see that the **non-transfer** agent ends at roughly the number of calls made by the **transfer** agent after 10,000 timesteps.

Fig. 5 shows subaction and exploration action calls to moving the hand relative to the block for the **non-transfer** case. The graph shows a typical developmental pattern for the agent where initially this action is called as an exploration action, and then later this action becomes a subaction of other actions. (This graph is shown to 80,000 timesteps for clarity, but the trend continues throughout learning.)

In Fig. 6 we see that the number of DBNs does not grow without bound. In the **non-transfer** case, the first peak in the number of DBNs comes from first deleting unnecessary DBNs for actions that are closed. The second peak comes from deleting DBNs that do not become plans after 100,000 timesteps. The **transfer** cases begins with some transferred DBNs, but it too does not grow without bound.

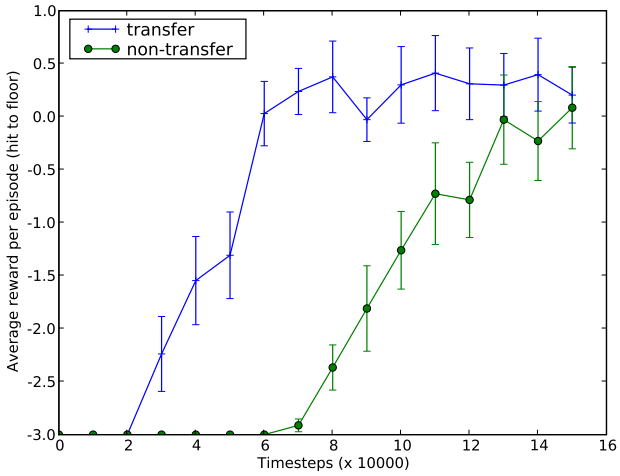

Fig. 3. The **transfer** agent using previously learned skills learns the task faster. Error bars are standard error.

VI. RELATED WORK

One way to see how an agent can reuse skills is to look at the transfer learning literature in reinforcement learning. One common method of transfer learning involves transferring a value function from one task to another [17]. If the state and action spaces of the two methods do not match, then a mapping needs to be specified (or learned) between the two. Transfer learning can also be done by learning a model and then reusing parts of the model for each new task as was done in [18]. Our method is also model based, but the model is built in a developmental context.

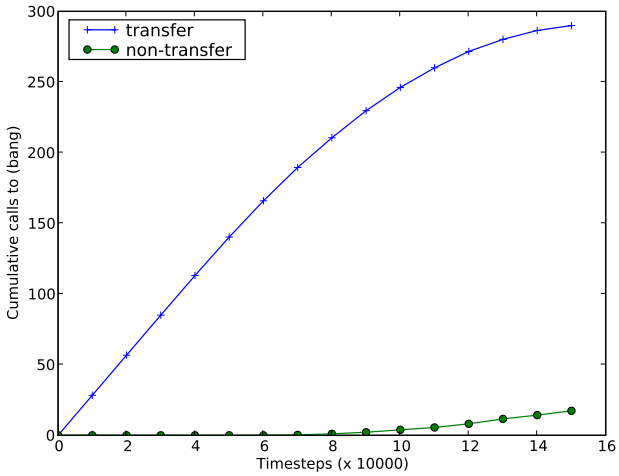

Fig. 4. The **transfer** agent makes more calls to set the variable *bang* to true because it is developmentally ready to learn that action.

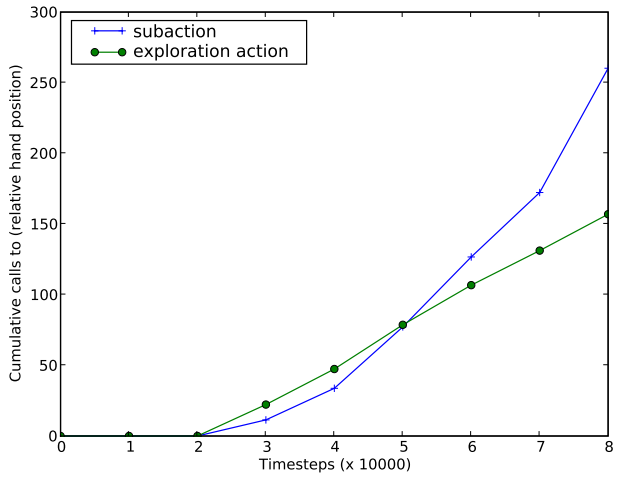

Fig. 5. The action of moving the hand relative to the block is first mostly called as an exploration action. But later, it is mostly called as a subaction for other actions. This graph is shown to 80,000 timesteps for clarity, but the trend continues throughout learning. (Graph made using the **non-transfer** agent.)

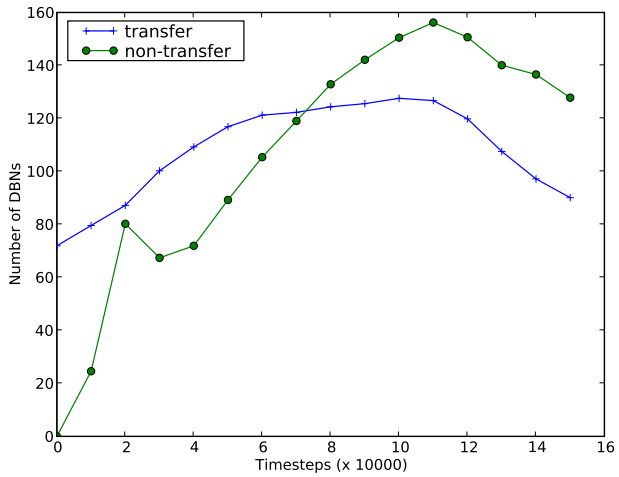

Fig. 6. The number of DBNs does not grow without bound.

Recently, there has been much work on model-based approaches that learn structure in the form of DBNs or decision trees. Degris *et al.* [2006] proposed a method called SDYNA that learns a structured representation and then uses that structure to compute a value function. Similarly, Strehl *et al.* [2007] learn a DBN to predict each component of a factored state MDP. Both of these methods are evaluated in discrete environments where transitions occur over one-timestep intervals. Another method is learning probabilistic planning rules [21]. In the domain of first-order logic they learn rules that given a context and an action provide a distribution over results. This algorithm also assumes a discrete state space and that the agent already has basic actions such as *pick up*.

Another approach is to learn subgoals to reach states. McGovern and Barto [2001] proposed a method whereby an agent autonomously finds subgoals based on bottleneck states that are visited often during successful trials and rarely during unsuccessful ones. Subgoals have also been found by constructing a transition graph based on recent experience and then searching for "access states" [23] that allow the agent to go from one partition of the graph to another. In Barto *et al.* [24] options are learned to achieve salient events. However, these salient events are determined outside the algorithm, and all of this work takes place in discrete environments.

VII. DISCUSSION AND FUTURE WORK

A developmental robot must be able to learn continuously throughout the long period of development. To do this, it must manage the complexity of the environment without overwhelming its resources. We can use the qualitative abstraction to reduce this complexity because we only pay the cost of the abstraction. But it is important to balance the gain and cost of the abstraction. This cost is increased by learning DBNs, their contexts, and landmarks. But we gain from the reduced complexity that comes from the hierarchical actions that these constructions support. In future work we will further investigate this balancing of abstractions.

A broader issue is that of learning a more sophisticated representation. There is an important limitation in the representation used by QLAP. In QLAP, events are represented as changes in the values of single variables. This means that the result of an action must already be represented in QLAP with a single variable. QLAP represents states this way because this enables it to do the initial learning.

To see an illustration of this limitation, consider the example of one block sitting on top of another block. How can this state be represented in QLAP? It must be represented as a conjunction of variables. For example, for two blocks b^1 and b^2, to represent the state that b^1 is on top of b^2 we would need the cumbersome representation

$$b^1_x = b^2_x \quad \wedge \quad b^1_y = b^2_y \quad \wedge \quad b^1_z > b^2_z \tag{2}$$

This formula abstracts many low-level states, and we would like to refer to all of those states using a single symbol. Using a single symbol, we can start to predict results of

being in this state or when this state will occur. We refer to this problem of creating a symbol to represent an important state as the *state abstraction problem*, and we call such states *high-level states*.

There is another representational issue, the *event abstraction problem* is the problem of abstracting time-series data into a single event. Consider the event of block b^1 falling off block b^2. This event takes more than one timestep and involves a change in multiple variables. We refer to such events as *high-level events*.

If we can define high-level states, then we can define high-level events as transitions between high-level states. If we do this, the set of possible solutions to the state representation problem is the set of ways to group variables (and variable values) into high-level states.

To aggregate these states we can consider both bottom-up and top-down approaches. One bottom-up method is to identify high-level states as those that are *stable*. We define states as stable as those that come up often and stay that way for a relatively long period of time. One block on top of another is stable, and two blocks sitting on a table is also stable. Another example is a block sitting on the floor after it has fallen off the table. Defining a high-level state as a block sitting on the floor allows us to define the high-level event of the block falling off the table without specifying an additional variable. The high-level event is the transition from the high-level state of the block being on the table to the high-level state of the block being on the floor.

While bottom up approaches will be necessary, we may also want to consider top-down approaches that consider the agent's current state. One top-down approach would be to look for "important" states that are associated with positive outcomes for the agent.

VIII. CONCLUSION

We present a demonstration of skill reuse in a continuous environment where no task is specified. The agent simultaneously learns a state abstraction and a set of task-general actions in a developmental progression. Because the agent only pays the cost of the abstraction, this is a small step towards being able to do continuous learning.

IX. ACKNOWLEDGMENTS

This work has taken place in the Intelligent Robotics Lab at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Intelligent Robotics lab is supported in part by grants from the Texas Advanced Research Program (3658-0170-2007), and from the National Science Foundation (IIS-0413257, IIS-0713150, and IIS-0750011). The authors would also like to thank Matthew Taylor, Jeremy Stober, Shilpa Gulati, and Changhai Xu for helpful discussions, comments and suggestions.

REFERENCES

[1] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen, "Autonomous mental development by robots and animals," *Science*, vol. 291, pp. 599–600, 2001.

[2] J. J. Gibson, *The Ecological Approach to Visual Perception*. New Jersey, USA: Lawrence Erlbaum Associates, 1979.

[3] A. Bonarini, A. Lazaric, and M. Restelli, "Incremental Skill Acquisition for Self-Motivated Learning Animats," *Lecture Notes in Computer Science*, vol. 4095, p. 357, 2006.

[4] C. M. Vigorito and A. G. Barto, "Autonomous hierarchical skill acquisition in factored mdps," in *Yale Workshop on Adaptive and Learning Systems*, New Haven, Connecticut, 2008.

[5] J. Mugan and B. Kuipers, "Autonomously learning an action hierarchy using a learned qualitative state representation," in *Proc. of the Int. Joint Conf. on Artificial Intelligence*, 2009.

[6] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," *Computational Intelligence*, vol. 5, no. 2, pp. 142–150, 1989.

[7] J. Mugan and B. Kuipers, "Learning to predict the effects of actions: Synergy between rules and landmarks," in *Proc. of the Int. Conf. on Development and Learning*, 2007.

[8] B. Kuipers, *Qualitative Reasoning*. Cambridge, Massachusetts: The MIT Press, 1994.

[9] C. Boutilier, R. Dearden, and M. Goldszmidt, "Exploiting structure in policy construction," in *IJCAI*, 1995, pp. 1104–1113.

[10] G. L. Drescher, *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, MA: MIT Press, 1991.

[11] J. Mugan and B. Kuipers, "Towards the application of reinforcement learning to undirected developmental learning," in *Proc. of the Int. Conf. on Epigenetic Robotics*, 2008.

[12] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge MA: MIT Press, 1998.

[14] J. Mugan and B. Kuipers, "A comparison of strategies for developmental action acquisition in QLAP," in *Proc. of the Int. Conf. on Epigenetic Robotics (under review)*, 2009.

[15] P. Oudeyer, F. Kaplan, and V. Hafner, "Intrinsic Motivation Systems for Autonomous Mental Development," *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 2, pp. 265–286, 2007.

[16] J. Klein, "Breve: a 3d environment for the simulation of decentralized systems and artificial life," in *Proc. of the Int. Conf. on Artificial Life*, 2003.

[17] M. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, no. 1, pp. 2125–2167, 2007.

[18] M. Taylor, N. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *Proceedings of the European Conference on Machine Learning and Principles and Pratice of Knowledge Discovery in Databases (ECML PKDD)*. Springer, 2008.

[19] T. Degris, O. Sigaud, and P. Wuillemin, "Learning the structure of factored Markov decision processes in reinforcement learning problems," in *ICML*, 2006, pp. 257–264.

[20] A. Strehl, C. Diuk, and M. Littman, "Efficient structure learning in factored-state MDPs," in *AAAI*, vol. 22, no. 1, 2007, p. 645.

[21] H. Pasula, L. Zettlemoyer, and L. Kaelbling, "Learning symbolic models of stochastic domains," *JAIR*, vol. 29, pp. 309–352, 2007.

[22] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *ICML*, 2001, pp. 361–368. [Online]. Available: citeseer.ist.psu.edu/mcgovern01automatic.html

[23] O. Simsek, A. Wolfe, and A. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," *ICML*, pp. 816–823, 2005.

[24] A. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," *ICDL*, 2004.