Few-Shot Imitation Learning with Disjunctions of Conjunctions of Programs

Tom Silver, Kelsey Allen, Leslie Kaelbling, Josh Tenenbaum
MIT, Cambridge, MA

Abstract

We describe an expressive class of policies that can be efficiently learned from a few demonstrations. Policies are represented as disjunctions (logical or's) of conjunctions (logical and's) of programs from a small domain-specific language (DSL). We define a prior over policies with a probabilistic grammar and derive an approximate Bayesian inference algorithm to learn policies from demonstrations. In experiments, we study five strategy games played on a 2D grid with one shared DSL. After a few (at most eight) demonstrations of each game, the inferred policies generalize to new game instances that differ substantially from the demonstrations. We also find that policies inferred from single demonstrations can be used for efficient exploration to dramatically reduce RL sample complexity.

1 Introduction

People are remarkably good at understanding and generalizing game tactics after only a few demonstrations. For example, consider the "Reach for the Star" game in Figure 1, where a purple agent must navigate to a yellow star in the presence of gravity. Clicking an empty cell creates a new blue object and clicking an arrow (bottom right) moves the agent. After only three demonstrations of the "stair building" tactic, it is easy to generalize to new and larger instances of the game. We are interested in similarly learning and generalizing policies from very few demonstrations.

There are three immediate requirements if we wish to learn a policy $\pi \in \Pi$ that generalizes from only a few demonstrations of an expert policy. (1) The policy class Π must be general and expressive enough to contain (a policy close to) the expert policy. (2) Our prior over polices $Pr(\pi)$ should encode any inductive bias that we have available. (3) An inference algorithm must be able to efficiently search Π to discover a π with high posterior probability $Pr(\pi|\mathcal{D}) \propto Pr(\mathcal{D}|\pi)Pr(\pi)$ from demonstrations $\mathcal{D}$.

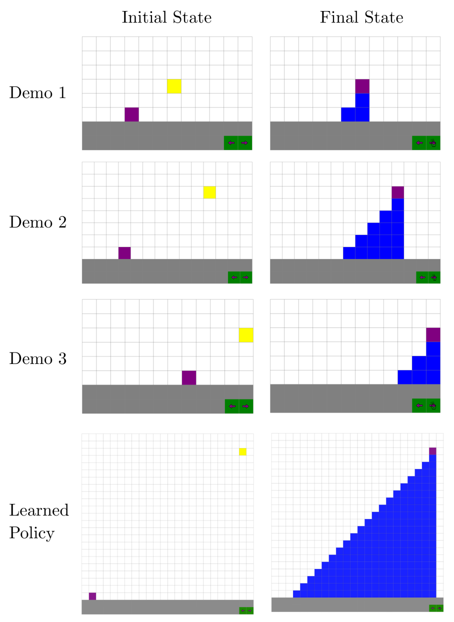


Figure 1: "Reach for the Star" game demonstrations and learned policy.

Policies implemented as programs drawn from a domain-specific language (DSL) can be very expressive (Wingate et al., 2013; Xu et al., 2018; Lázaro-Gredilla et al., 2019). A probabilistic grammar can be used to define a structured prior over programs (Manning et al., 1999; Goodman et al., 2008; 2014; Piantadosi et al., 2016). However, despite great progress in program induction (Lau, 2001; Liang et al., 2010; Gulwani, 2011; Menon et al., 2013), inference still requires domain-specific approximations (Lake et al., 2015) or enumeration with learned guidance (Balog et al., 2017; Devlin et al., 2017; Ellis et al., 2018a;b; Lázaro-Gredilla et al., 2019). If the DSL is restricted to involve only logical combinations of predicates from a finite set, e.g., disjunctions of conjunctions (Valiant, 1985; Dietterich & Michalski, 1986; Haussler, 1988; Dechter & Mateescu, 2007; Kansky et al., 2017), then many Boolean learning algorithms become available

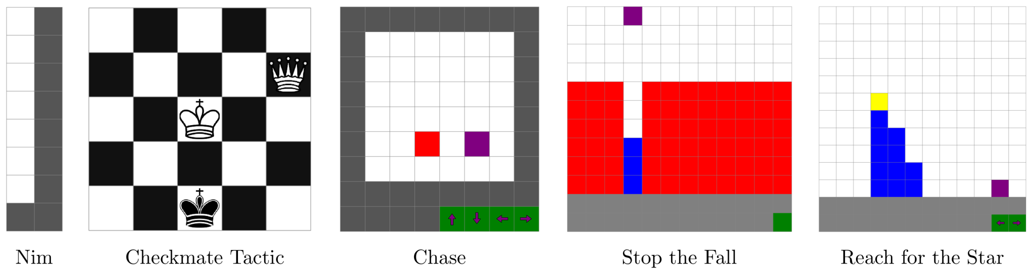


Figure 2: The five strategy games studied in this work. See the appendix for descriptions and additional illustrations.

(Quinlan, 1986; Mitchell, 1978; Wang et al., 2017), but at the expense of expressivity. Our main insight is that a DSL can be combined with Boolean logic to create a policy class that enjoys the generality of the DSL, the efficient learnability of Boolean logic, and the natural structured priors of both. We call this class `DCP` as it involves disjunctions of conjunctions of programs.

The development of `DCP` was inspired by our study of five strategy games played on a 2D grid of varying size (see Figure 2). These games are diverse in their transition rules and in the tactics required to win, but they share a common state space ($\mathcal{S} = \bigcup_{H=1}^{\infty} \bigcup_{W=1}^{\infty} \{1, 2, ..., V\}^{H \times W}$; variable-sized grids with discrete-valued cells) and action space ($\mathcal{A} = \mathbb{Z}^2$; single "clicks" on any cell). We can thus build our policies for all five games from one shared, small DSL. Our DSL is inspired by how a human might focus and shift their attention between cells when analyzing a scene (Ullman, 1987; Agre & Chapman, 1987; Duncan & Humphreys, 1989; Ballard et al., 1997; Hay et al., 2018). In experiments, we learn policies in `DCP` from a few demonstrations in each game. We find that the learned policies generalize perfectly to test game instances after at most eight demonstrations. We also show that we can start with a single demonstration and quickly improve an imperfect inferred policy with Q-learning. Overall, our experiments suggest that `DCP` is an efficient, flexible approach for learning rich, generalizable policies from very little data.

2 Related Work

Imitation learning is the problem of learning and generalizing from expert demonstrations (Pomerleau, 1991; Schaal, 1997; Abbeel & Ng, 2004). To cope with limited data, e.g., due to robotic resource constraints (Billard et al., 2008; Argall et al., 2009; Konidaris et al., 2012), the demonstrations can be used in combination with additional reinforcement learning (Hester et al., 2018; Nair et al., 2018). Alternatively, a mapping from demonstrations to policies can be learned from a background set of tasks (Duan et al., 2017; Finn et al., 2017). A third option is to introduce a prior over a structured class of policies (Andre & Russell, 2002; Doshi-velez et al., 2010; Wingate et al., 2011), e.g., hierarchical or compositional policies (Niekum, 2013; Ranchod et al., 2015; Daniel et al., 2016; Krishnan et al., 2017). Our work here fits into the third tradition: our main contribution is a new policy class with a structured prior that enables efficient imitation learning. We also extend the first tradition in our Q-learning experiments.

We define policies in terms of programs (Andre & Russell, 2002; Wingate et al., 2013; Xu et al., 2018) and a prior over policies using a probabilistic grammar (Goodman et al., 2008; 2014; Piantadosi et al., 2016). Similar priors appear in Bayesian concept learning from cognitive science (Tenenbaum, 1999; Tenenbaum & Griffiths, 2001; Lake et al., 2015; Ellis et al., 2018a;b). Incorporating insights from program induction for planning has seen renewed interest in the past year. Of particular note is the work by Lázaro-Gredilla et al. (2019), who learn object manipulation concepts from before/after image pairs that can be transferred between 2D simulation and a real robot. They define a DSL that involves visual perception with shifting attention, working memory, new object imagination, and object manipulation. We use a DSL that similarly involves shifting attention and object-based (grid cell-based) actions. However, in this work, we focus on the problem of *efficient inference* for learning programs which specify policies, and make use of full demonstrations of a policy (instead of start/end configurations alone). We compare our inference method against enumeration in experiments.

Method	Input	Output	Description
shift(p, dp)	$(\mathbb{Z}^2, \mathbb{Z}^2)$	$\mathbb{Z}^2$	Shift attention from p to $(p + dp)$
cell_is_value(v, p, s)	$(\mathbb{N}, \mathbb{Z}^2, \mathcal{S})$	$\{0, 1\}$	Dereference p in s and compare with v
find_match(v, s)	$(\mathbb{N}, \mathcal{S})$	$\mathbb{Z}^2 \cup \{\emptyset\}$	Find a cell in s with value v if one exists
do_until(δ, c_1, c_2, p, s)	See text	$\{0, 1\}$	See text

Table 1: Methods comprising the domain-specific language (DSL) used in this work.

3 The DCP Policy Class

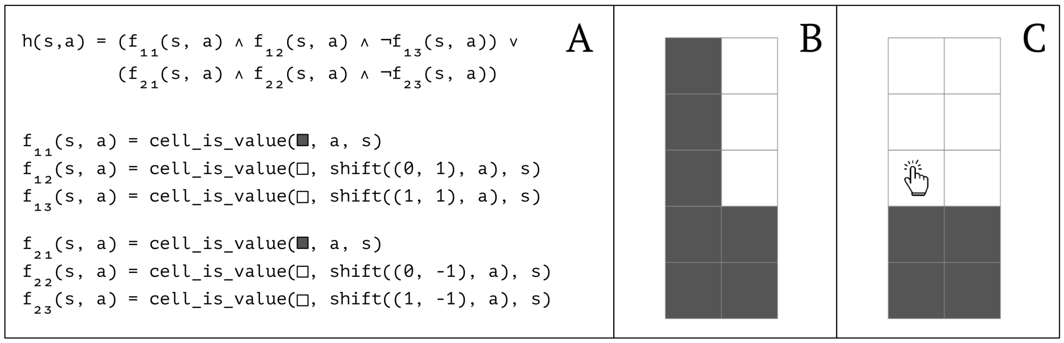
Here we define the DCP (Disjunctions of Conjunctions of Programs) policy class. Policies $\pi \in$ DCP are stochastic; given a state $s \in \mathcal{S}$, $\pi(a|s)$ is a distribution over actions $a \in \mathcal{A}$. We restrict our attention to discrete state and action spaces throughout this work. The distribution $\pi(a|s)$ is determined by a Boolean function $h_\pi : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$ so that $\pi(a|s) \propto h_\pi(s, a)$. For a given s, there will often be one a such that $h_\pi(s, a) = 1$ (in which case $\pi(a|s)$ is deterministic), but there may also be multiple or no such actions. The Boolean function h_π decomposes into a disjunction of conjunctions of other Boolean functions:

$$h_\pi(s, a) \triangleq (f_{1,1}(s, a) \wedge ... \wedge f_{1,n_1}(s, a)) \vee ... \vee (f_{m,1}(s, a) \wedge ... \wedge f_{m,n_m}(s, a))$$

where each function $f_{i,j}(s, a)$ is a (possibly negated) *program* drawn from a DSL.

The specific DSL we use in this work is inspired by early work in visual routines (Ullman, 1987) and deictic references (Agre & Chapman, 1987; Ballard et al., 1997). Each program $f_{i,j}$ implements a procedure for shifting an "attention" pointer from one grid cell to another depending on the value of the currently attended cell. Given input (s, a), the attention pointer is initialized to the grid cell in s associated with action a. There are only four methods in the DSL; they are summarized in Table 1. The one non-atomic method is do_until, which repeatedly shifts an initial attention pointer p following a shift routine δ until one of two conditions c_1, c_2 are met; if c_1 is met first, the method returns 1; otherwise it returns 0. The conditions c_1, c_2 can be defined using cell_is_value or another do_until.

See Figure 3 for a complete example of a policy in DCP using the DSL described above.



```
h(s,a) = (f11(s, a) ∧ f12(s, a) ∧ ¬f13(s, a)) ∨
         (f21(s, a) ∧ f22(s, a) ∧ ¬f23(s, a))

f11(s, a) = cell_is_value(■, a, s)
f12(s, a) = cell_is_value(□, shift((0, 1), a), s)
f13(s, a) = cell_is_value(□, shift((1, 1), a), s)

f21(s, a) = cell_is_value(■, a, s)
f22(s, a) = cell_is_value(□, shift((0, -1), a), s)
f23(s, a) = cell_is_value(□, shift((1, -1), a), s)
```

Figure 3: Example of a policy in DCP for the "Nim" game. (A) $h(s, a)$ is a disjunction of conjunctions of programs from a DSL. The induced policy is $\pi(a|s) \propto h(s, a)$. (B) Given state s, (C) there is one action selected by h. This policy encodes the "leveling" tactic, which wins the game.

4 Imitation Learning as Bayesian Inference

We now describe an approximate Bayesian inference algorithm for imitation learning of policies in DCP from demonstrations $\mathcal{D}$. We will define the prior $Pr(\pi)$ and the likelihood $Pr(\mathcal{D}|\pi)$ and then describe an approximate inference algorithm for the posterior $Pr(\pi|\mathcal{D}) \propto Pr(\mathcal{D}|\pi)Pr(\pi)$. We can then use a posterior over policies to derive a state-conditional posterior over actions, giving us a final stochastic policy.

4.1 Policy Prior $Pr(\pi)$ and Likelihood $Pr(\mathcal{D}|\pi)$

Recall that a policy $\pi \in \texttt{DCP}$ is defined in terms of Boolean programs $f_{i,j}$ from a DSL. We factor the prior into a product of priors over these programs: $Pr(\pi) \propto \prod_{i=1}^{M} \prod_{j=1}^{N_i} Pr(f_{i,j})$. If there were a finite number of programs and their prior was uniform, the policy prior would favor policies with the fewest programs (Rissanen, 1978). But we are instead in the regime where there are an infinite number of programs and some are much simpler than others. We thus define a prior over programs using a probabilistic grammar (Manning et al., 1999). Given a probabilistic grammar, it is straightforward to enumerate programs with monotonically decreasing probability via best-first search. We take advantage of this property during inference. The specific grammar we use in this work (Table 2 in the appendix) assigns a uniform distribution over the rules associated with each head.

We now describe the likelihood $Pr(\mathcal{D}|\pi)$. Let $(s_0, a_0, ..., s_{T-1}, a_{T-1}, s_t)$ be a demonstration in $\mathcal{D}$. The probability of this demonstration given π is $Pr(s_0) \prod_{t=0}^{T-1} Pr(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$ where $Pr(s_{t+1}|s_t, a_t)$ is the (unknown) transition distribution. As a function of π, the likelihood of the demonstration is thus proportional to $\prod_{t=0}^{T-1} \pi(a_t|s_t)$. Given N i.i.d. demonstrations $\mathcal{D}$, the likelihood is then $Pr(\mathcal{D}|\pi) \propto \prod_{i=1}^{NT} \pi(a_i|s_i)$.

4.2 Approximating the Posterior $Pr(\pi|\mathcal{D})$

We now have the prior $Pr(\pi)$ and the likelihood $Pr(\mathcal{D}|\pi)$ and we wish to compute the posterior $Pr(\pi|\mathcal{D}) \propto Pr(\mathcal{D}|\pi)Pr(\pi)$. Recall that a policy π is determined by a function h_π which takes a state-action pair (s, a) as input and produces a Boolean output indicating whether a should be taken by π in state s. We can therefore reduce the problem of policy learning to classification. Positive examples are supplied by the demonstration; negative examples can be derived by considering actions *not* taken from the states in the demonstration. This reduction to classification is approximate; if multiple actions are permissible by the expert policy, this classification dataset will include actions mislabelled as negative.

Recall further that h_π is represented as a disjunction of conjunctions of programs $f_{i,j}$ drawn from a DSL, where each program is a mapping from state-actions to Booleans. Given a finite set of programs, we can run each program $f_{i,j}$ over all positive and negative state-actions to create Boolean feature vectors, where each feature in each vector corresponds to the output of one program on one state-action pair. We then have a binary classification problem with multidimensional Boolean inputs. The problem of learning a binary classifier as a disjunction of conjunctions of Booleans is very well understood (Mitchell, 1978; Valiant, 1985; Quinlan, 1986; Dietterich & Michalski, 1986; Haussler, 1988; Dechter & Mateescu, 2007; Wang et al., 2017). A fast approximate method that we use in this work is greedy decision tree learning. This method tends to learn small classifiers, which are hence likely according to our prior. From a learned classifier, we can derive the disjunction of conjunctions of programs h_π and the corresponding policy π.

We thus have a method to derive a single maximum-likelihood policy in DCP when the DSL is finite. To infer $Pr(\pi|\mathcal{D})$, we instead need to derive a posterior distribution over *many* policies with an *infinite* DSL and a *prior* over programs. A simple extension of the above method is to learn an ensemble of classifiers rather than a single one. We then weight each member of the ensemble according to the (unnormalized) posterior. This gives us a particle-based approximate posterior over many policies. To address the remaining challenge of an infinite DSL, we start with an empty set of programs and iteratively add new ones by searching our probabilistic grammar in best-first order. After each batch of new programs, we produce a new set of particles using the extension above and collect a cumulative particle set to represent an overall approximate posterior. We can stop after a fixed number of iterations or after the enumerated program prior probabilities fall below a threshold. See Algorithm 1 in the appendix for a summary of the overall method.

5 Experiments and Results

We study five diverse strategy games (Figure 2) that share a common state space ($\mathcal{S} = \bigcup_{H=1}^{\infty} \bigcup_{W=1}^{\infty} \{1, 2, ..., V\}^{H \times W}$; variable-sized grids with discrete-valued cells) and action space

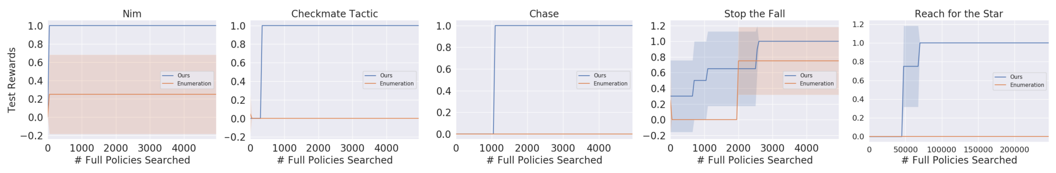


Figure 4: Test environment rewards accrued as imitation learning progresses. Our learning method efficiently discovers policies that generalize from the demonstrations. Enumerating full policies is intractable for even the simplest games. Rewards are averaged over four test environments.

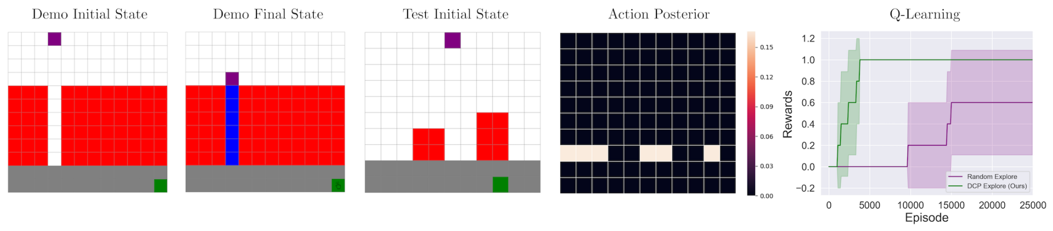


Figure 5: Learning from one demonstration combined with RL in "Stop the Fall." One demonstration is given and a policy distribution is inferred; the resulting action posterior gives significant weight to six actions in the initial state. Q-learning in the test environment converges faster when this learned posterior is used to guide exploration. Results are averaged over five random seeds.

($\mathcal{A} = \mathbb{Z}^2$; single "clicks" on any cell). Winning the games can require arbitrarily long sequences of actions. Winning each game results in a reward of 1, losing in 0. See appendix for details.

5.1 Learning and Generalizing from Demonstrations

Here we evaluate the extent to which learned policies in DCP generalize beyond given demonstrations. For each of the five games, we start with demonstrations $\mathcal{D}$ and infer a posterior over policies $Pr(\pi|\mathcal{D})$ as described in Section 4. Given a new state s, we compute a posterior over actions $Pr(a|s) \propto \sum_{\pi} \pi(a|s)Pr(\pi|\mathcal{D})$. We use the top 25 particles from the approximate policy prior to compute the action posterior. We then execute the MAP action $\arg\max_{a\in\mathcal{A}} Pr(a|s)$. All demonstrations and tests are illustrated in the appendix. We find that at most eight demonstrations are required to generalize to the test games. Figure 4 shows test time rewards as a function of the number of policies produced over time by our method versus full policy enumeration. Our method efficiently discovers policies that generalize; enumerating full policies is intractable for all games.

5.2 Q-Learning from One Demonstration

Now we examine whether imitation learning from a single demonstration can be combined with Q-learning to arrive at an optimal policy with lower sample complexity than RL alone. We start by inferring a policy from the available demonstration and transferring to a new game instance. We then do tabular Q-learning in the new (fixed) game. During Q-learning, we select actions from an exploration policy – either the initial inferred policy or a random policy – with probability $\epsilon = 0.1$. We find that using the inferred policy for exploration can substantially reduce sample complexity (Figure 5). See the appendix for more details.

6 Conclusion

We have described an expressive class of policies and an approximate Bayesian method for learning them from a few demonstrations. Our work is a testament to the importance of inductive bias in reinforcement learning domains. If a small amount of bias can be codified in a DSL, and if the resulting policy class can be efficiently searched, we can learn and generalize from far less data than would otherwise be possible.

Acknowledgments

We thank Anurag Ajay and Ferran Alet for helpful discussions and feedback on earlier drafts. We gratefully acknowledge support from NSF grants 1523767 and 1723381; from ONR grant N00014-13-1-0333; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from Honda Research; from the MIT-Sensetime Alliance on AI. and from the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216. KA acknowledges support from NSERC. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Philip E Agre and David Chapman. Pengi: An implementation of a theory of activity. *AAAI Conference on Artificial Intelligence*, 1987.

David Andre and Stuart J Russell. State abstraction for programmable reinforcement learning agents. In *AAAI Conference on Artificial Intelligence*, 2002.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Dana H Ballard, Mary M Hayhoe, Polly K Pook, and Rajesh PN Rao. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, 20(4):723–742, 1997.

Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. *International Conference on Learning Representations*, 2017.

Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. pp. 1371–1394. Springer, 2008.

Christian Daniel, Herke Van Hoof, Jan Peters, and Gerhard Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, 2016.

Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artificial intelligence*, 171(2-3):73–106, 2007.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. *International Conference on Machine Learning*, 2017.

Thomas G Dietterich and Ryszard S Michalski. Learning to predict sequences. *Machine learning: An artificial intelligence approach*, 2, 1986.

Finale Doshi-velez, David Wingate, Nicholas Roy, and Joshua B. Tenenbaum. Nonparametric bayesian policy priors for reinforcement learning. In *Advances in Neural Information Processing Systems*. 2010.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *Advances in neural information processing systems*, 2017.

John Duncan and Glyn W Humphreys. Visual search and stimulus similarity. *Psychological review*, 96(3):433, 1989.

Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. Learning libraries of subroutines for neurally–guided bayesian program induction. *Advances in Neural Information Processing Systems*, 2018a.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. *Advances in Neural Information Processing Systems*, 2018b.

Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *Conference on Robot Learning*, 2017.

Noah D Goodman, Joshua B Tenenbaum, Jacob Feldman, and Thomas L Griffiths. A rational analysis of rule-based concept learning. *Cognitive science*, 32(1):108–154, 2008.

Noah D Goodman, Joshua B Tenenbaum, and Tobias Gerstenberg. Concepts in a probabilistic language of thought. Technical report, Center for Brains, Minds and Machines (CBMM), 2014.

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM SIGPLAN Notices*, 46(1):317–330, 2011.

David Haussler. Quantifying inductive bias: AI learning algorithms and valiant's learning framework. *Artificial intelligence*, 36(2):177–221, 1988.

Nicholas Hay, Michael Stark, Alexander Schlegel, Carter Wendelken, Dennis Park, Eric Purdy, Tom Silver, D Scott Phoenix, and Dileep George. Behavior is everything–towards representing concepts with sensorimotor contingencies. *AAAI Conference on Artificial Intelligence*, 2018.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, 2018.

Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *International Conference on Machine Learning*, 2017.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3): 360–375, 2012.

Sanjay Krishnan, Roy Fox, Ion Stoica, and Ken Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. *Conference on Robot Learning*, 2017.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Tessa Lau. *Programming by demonstration: a machine learning approach*. PhD thesis, University of Washington, 2001.

Miguel Lázaro-Gredilla, Dianhuan Lin, J. Swaroop Guntupalli, and Dileep George. Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *Science Robotics*, 4(26), 2019.

Percy Liang, Michael I Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. *International Conference on Machine Learning*, 2010.

Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

Aditya Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, and Adam Kalai. A machine learning framework for programming by example. *International Conference on Machine Learning*, 2013.

Tom Michael Mitchell. Version spaces: an approach to concept learning. Technical report, Stanford University, 1978.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *International Conference on Robotics and Automation*, pp. 6292–6299. IEEE, 2018.

Scott D Niekum. *Semantically grounded learning from unstructured demonstrations*. PhD thesis, University of Massachusetts, Amherst, 2013.

Steven T Piantadosi, Joshua B Tenenbaum, and Noah D Goodman. The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological review*, 123(4):392, 2016.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Pravesh Ranchod, Benjamin Rosman, and George Konidaris. Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. *International Conference on Intelligent Robots and Systems*, 2015.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 1997.

Joshua B Tenenbaum and Thomas L Griffiths. Generalization, similarity, and bayesian inference. *Behavioral and brain sciences*, 24(4):629–640, 2001.

Joshua Brett Tenenbaum. *A Bayesian framework for concept learning*. PhD thesis, Massachusetts Institute of Technology, 1999.

Shimon Ullman. Visual routines. pp. 298–328. Elsevier, 1987.

Leslie G Valiant. Learning disjunction of conjunctions. *International Joint Conference on Artificial Intelligence*, 1985.

Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. A bayesian framework for learning rule sets for interpretable classification. *The Journal of Machine Learning Research*, 18(1):2357–2393, 2017.

David Wingate, Noah D Goodman, Daniel M Roy, Leslie P Kaelbling, and Joshua B Tenenbaum. Bayesian policy search with policy priors. *International Joint Conference on Artificial Intelligence*, 2011.

David Wingate, Carlos Diuk, Timothy O’Donnell, Joshua Tenenbaum, and Samuel Gershman. Compositional policy priors. Technical report, Massachusetts Institute of Technology, 2013.

Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *International Conference on Robotics and Automation*, 2018.

7 APPENDIX

7.1 PROBABILISTIC GRAMMAR FOR DSL

Symbol	Substitution	Probability
START	CONDITION + "$p, s)$"	0.25
START	CONDITION + SHIFT + "$p, s)$"	0.25
START	CONDITION + FIND_MATCH + "$p, s)$"	0.25
START	CONDITION + SHIFT + FIND_MATCH + "$p, s)$"	0.25
CONDITION	VALUE_CONDITION	0.5
CONDITION	DO_UNTIL_CONDITION	0.5
FIND_MATCH	"find_match(" +VALUE + ","	1.0
SHIFT	"shift(" +DIRECTION + ","	1.0
VALUE_CONDITION	"cell_is_value(" +VALUE + ","	1.0
DO_UNTIL_CONDITION	"do_until(λx :" +SHIFT + "x)," + "$\lambda x, y$:" +CONDITION + "x, y)," + "$\lambda x, y$:" +CONDITION + "x, y)"	1.0
DIRECTION	"(-1, 0)" | "(1, 0)" | "(0, -1)" | "(0, 1)"	$\frac{1}{9}$ (each)
DIRECTION	"(-1, 1)" | "(1, -1)" | "(1, -1)" | "(-1, 1)"	$\frac{1}{9}$ (each)
DIRECTION	SHIFT + DIRECTION+)	$\frac{1}{9}$
VALUE	"1" | "2" | ... | "V"	$\frac{1}{\|V\|}$ (each)

Table 2: The probabilistic grammar used to define a prior over programs from the DSL (see Table 1). All substitution probabilities are uniform per symbol. The number of values varies per game ($3 \leq V \leq 9$).

7.2 Approximate Bayesian Inference Pseudocode

Algorithm 1: Imitation learning as approximate Bayesian inference

```
input: Demonstrations $\mathcal{D}$, ensemble size $K$
/* Initialize empty lists                                              */
state_actions, X, y, programs, program_priors, policy_particles, weights;
/* Compute all binary labels                                           */
for $(s, a^*) \in \mathcal{D}$ do
    for $a \in \mathcal{A}$ do
        state_actions.append($(s, a)$);
        X.append([]);
        $\ell = (a == a^*)$ ;        // Label whether action was taken by expert
        y.append($\ell$);
    end
end
/* Iteratively grow the program set                                    */
while True do
    program, prior = generate_next_program() ;            // Search the grammar
    programs.append(program);
    program_priors.append(prior);
    for $i$ in $1, ..., |state\_actions|$ do
        $(s, a)$ = state_actions[i];
        x = program($s, a$) ;           // Apply program to get binary feature
        X[i].append(x);
    end
    /* See Algorithm 2                                                 */
    particles, iter_weights = infer_disjunctions_of_conjunctions(X, y, program_priors, $K$);
    policy_particles.extend(create_policies(particles, programs));
    weights.extend(iter_weights);
    /* Stop after max iters or program prior threshold is met          */
    if stop_condition() then
        return policy_particles, weights;
    end
end
```

Algorithm 2: Inferring disjunctions of conjunctions (Algorithm 1 subroutine)

```
input: Feature vectors $X$, binary labels $y$, feature priors $p$, ensemble size $K$
/* Initialize empty lists                                              */
particles, weights;
for seed in $1, ..., K$ do
    tree = DecisionTree($seed$) ;              // Seed randomizes feature order
    tree.fit(X, y);
    /* Search the learned tree from root to True leaves                */
    h = convert_tree_to_dnf(tree);
    likelihood = get_likelihood(X, y, h);
    prior = get_prior(h, p);
    weight = likelihood $*$ prior;
    particles.append(h);
    weights.append(weight);
end
return particles, weights;
```

7.3 Environment Descriptions

In **Nim**, there are two columns (piles) of gray and white cells. Clicking on a gray cell changes all cells above and including the clicked cell to white; clicking on a white cell has no effect. After each gray cell click, a second player takes a turn, selecting another gray cell. The second player is modeled as part of the environment transition and plays optimally. When there are multiple optimal moves, one is selected randomly. The objective is to remove the *last* gray cell. The winning tactic is to "level" the columns by selecting the gray cell next to a white cell and diagonally up from another gray cell. Winning the game requires perfect play.

The **Checkmate Tactic** game is inspired by a common checkmating pattern in Chess. Note that only three pieces are involved in this game (two kings and a white queen) and that the board size may be $H \times W$ for any H, W, rather than the standard 8×8. Initial states in this game feature the black king somewhere on the boundary of the board, the white king two cells adjacent in the direction away from the boundary, and the white queen attacking the cell in between the two kings. Clicking on a white piece (queen or king) *selects* that piece for movement on the next action. Note that a selected piece is a distinct value from a non-selected piece. Subsequently clicking on an empty cell moves the selected piece to that cell if that move is legal. All other actions have no effect. If the action results in a checkmate, the game is over and won; otherwise, the black king makes a random legal move. The winning tactic selects the white queen and moves it to the cell between the kings.

Chase features a purple agent, a red adversary, gray walls, and four arrow keys. At each time step, the adversary randomly chooses a move (up, down, left, or right) that increases its distance from the agent. Clicking an arrow key moves the agent in the corresponding direction. Clicking a gray wall has no effect other than advancing time. Clicking an empty cell creates a new (blue) wall. The agent and adversary cannot move through gray or blue walls. The objective is to "catch" the adversary, that is, move the agent into the same cell. It is not possible to catch the adversary without creating a new wall; the adversary will always be able to move away before capture. The winning tactic advances time until the adversary reaches a corner, then builds a new wall next to the adversary so that it is trapped on three sides, then moves the agent to the adversary.

Stop the Fall involves a purple main object, gray static objects, red "lava", and a green button that turns on gravity and causes the purple object and blue objects to fall. Clicking an empty cell creates a blue object. The game is won when gravity is turned on and the purple object falls to rest without touching (being immediately adjacent to) lava. Thus the winning tactic requires building a stack of blue objects below the purple object that is high enough to prevent contact with lava, and then clicking the green button.

In **Reach for the Star**, a purple agent must move from the gray static floor to a cell with a yellow star. Left and right arrow keys cause the agent to move. Clicking on an empty cell creates a dynamic blue object. Gravity is always on, so blue objects fall until they are supported by a gray or another blue object. If the purple agent is adjacent to a blue object and the cell above the blue object is empty, the agent will move on top of the blue object when the corresponding arrow key is clicked. (In other words, the agent can only climb one object, not two or more.) The winning tactic requires building stairs between the star and agent and then moving the agent up them.

7.4 Experiment: Learning and Generalizing from Demonstrations

7.4.1 Experiment Details

Here we describe the details for the learning and generalization from demonstrations experiments presented in Section 5.1. We consider two methods for policy search: Bayesian Tactic Learning (ours) and full policy enumeration (baseline). Both methods iteratively generate new policies. As described in the main text, our method produces new policies by generating new programs and learning a disjunctions-of-conjuctions classifier on top of them. We use ensembles of 5 decision trees for all experiments.

Full policy enumeration generates new policies by explicitly searching over disjunctions of conjunctions of policies. We implement this search by creating a larger probabilistic grammar with disjunctions and conjunctions as part of the grammar; then we use the same best-first search machinery used by our method. Our grammar is such that the number of disjunctions and conjunctions obeys a geometric distribution ($p = 0.5$).

We give the same 4-8 game demonstrations to both methods. For each policy generated over time, we compute the (unnormalized) posterior conditioned on the demonstrations. We then save the

Game	**Clauses**	**Method Calls**
Nim	2	12
Checkmate Tactic	5	41
Chase	12	184
Stop the Fall	4	33
Reach for the Star	4	78

Table 3: Number of clauses (conjunctions) and method calls in MAP policies.

policy with the best posterior seen so far and run it on the test environments to produce the plots in Figure 4.

7.4.2 Additional Results

All demonstrations and learned policy performance on the test tasks are illustrated in Figures 6, 7, 8, 9, 10.

The MAP policy for Nim is illustrated below. It involves one more method call than the shortest optimal program shown in Figure 3. We summarize the other MAP programs in Table 3.

```
h = lambda p, s: (
    (
        not (cell_is_value('white', p, s)) and
        cell_is_value('white', shift((0, 1), p), s) and
        not (cell_is_value('white', shift((1, 1), p), s))
    )
)
or
(
    (
        not (cell_is_value('white', p, s)) and
        not (cell_is_value('white', shift((0, 1), p), s)) and
        cell_is_value('white', shift((0, -1), p), s) and
        not (cell_is_value('white', shift((1, -1), p), s))
    )
)
```

We tried many variations on the full policy enumeration search over disjunctions and conjunctions, including settings of p that incorporate our knowledge about the desired policies. We also tried small subsets of the DSL that we knew to be important for individual games. Quick calculations confirm that the policies required by the games are fundamentally difficult to find by enumeration. For example, the Nim policy illustrated in Figure 3 involves six short programs. The number of policies with this same description length is more than 100 million. In contrast, our method finds a winning policy for Nim after fewer than 100 iterations.

7.5 Experiment: Q-Learning from One Demonstration

Here we describe the details for the Q-learning from one demonstration experiment presented in Section 5.2. We start with the single demonstration of "Stop the Fall" illustrated in Figure 5 and learn a policy posterior with our method. We then enter a new instance of "Stop the Fall" where the action posterior does not perfectly transfer. In this new fixed instance, we perform tabular Q-learning ($\gamma = 0.95, \alpha = 0.5$). For each training episode, at each time step, we select an exploratory action with probability $\epsilon = 0.1$. As a baseline, we consider the standard ϵ-greedy approach of selecting an exploratory action uniformly at random. We then consider selecting exploratory actions by sampling from the original action posterior. For this approach, we sample an action from this posterior with probability $\epsilon_2 = 0.5$ and sample a uniformly random action otherwise. (Thus the overall probability of following the action posterior is $\epsilon\epsilon_2 = 0.05$.) The initial action distribution can be seen as sharply "narrowing down" the space of actions that should be explored. We find that this leads to sample complexity improvements, which matches intuition.

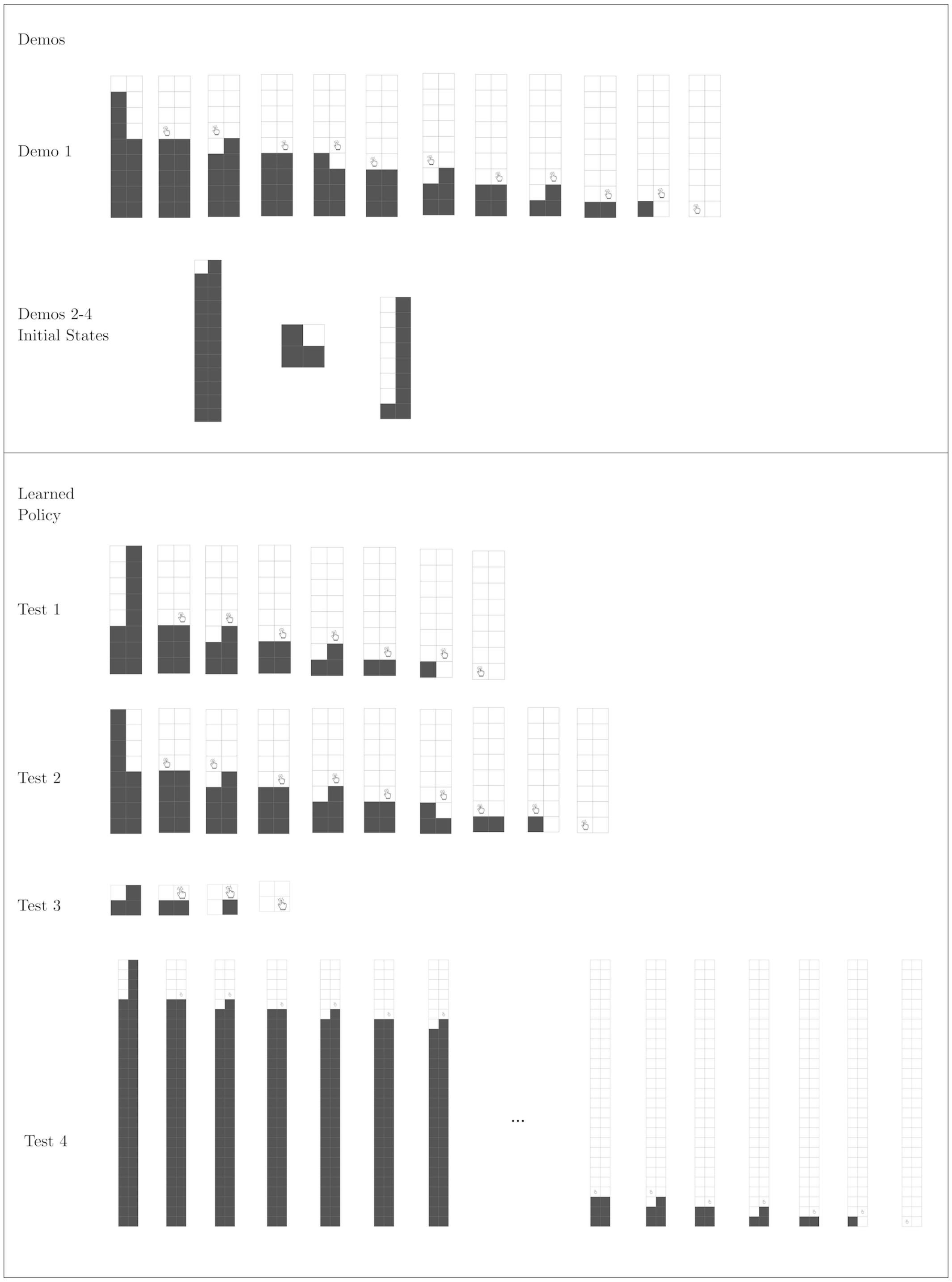


Figure 6: Demonstrations and generalization tests for the “Nim” game. After four demonstrations (top), the learned policy generalizes perfectly to the four tests (bottom), including one (Test 4) with a much larger grid than seen in the demonstrations. Note that player 2 moves are shown as separate observations for illustrative purposes but are not seen by the algorithm.

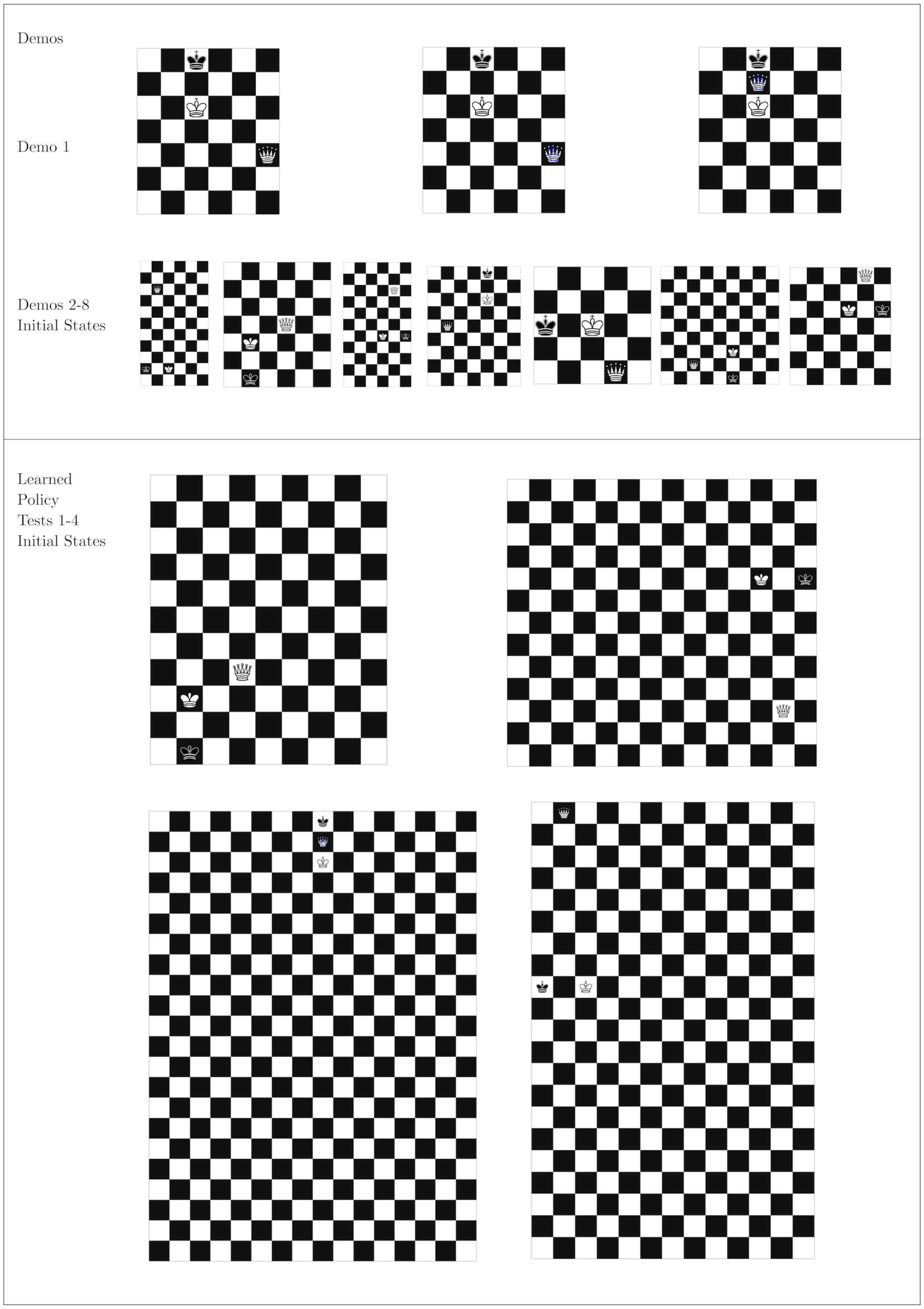


Figure 7: Demonstrations and generalization tests for the "Checkmate Tactic" game. After eight demonstrations (top), the learned policy generalizes perfectly to the four tests (bottom), including tests with much larger grids than seen in the demonstrations.

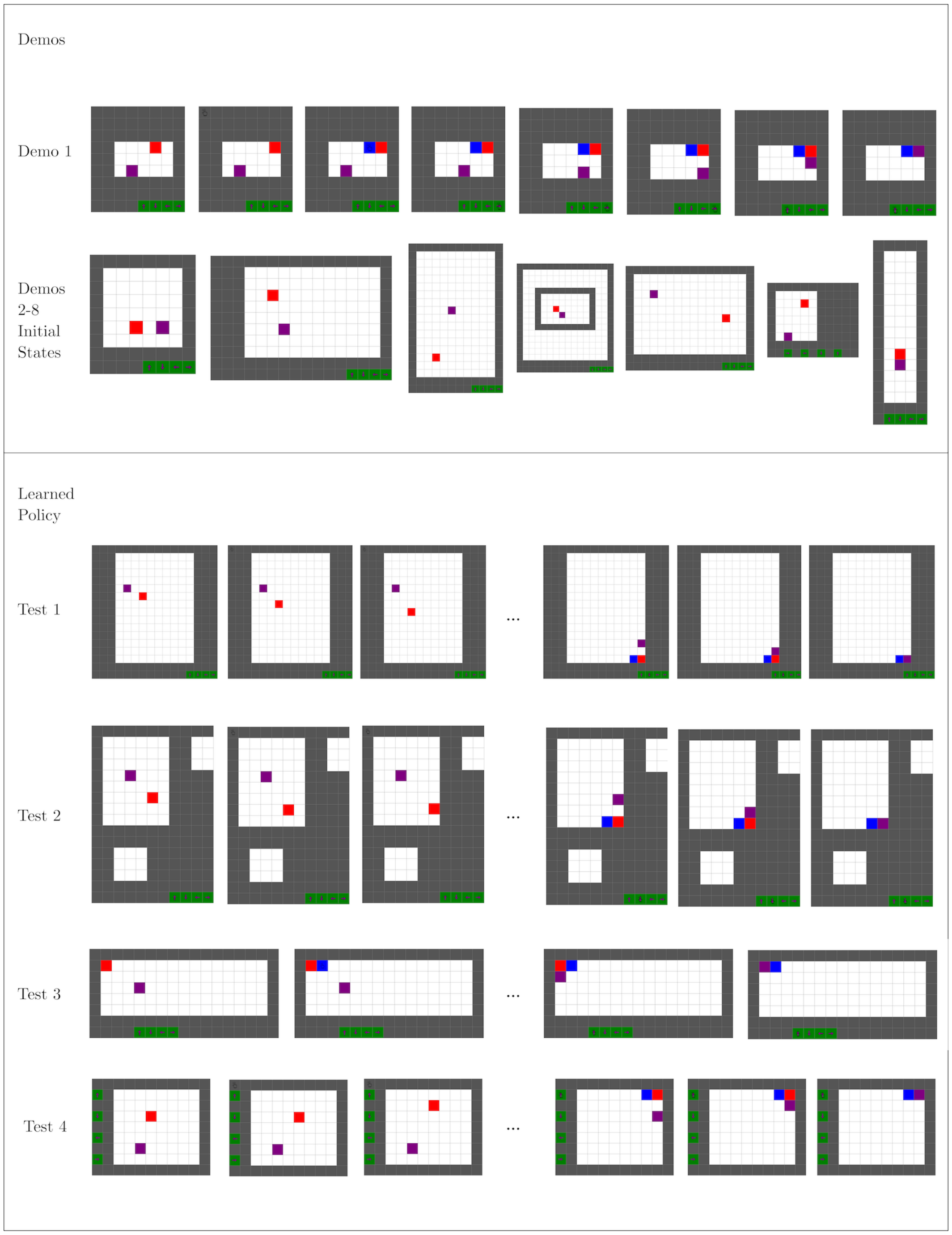


Figure 8: Demonstrations and generalization tests for the "Chase" game. After eight demonstrations (top), the learned policy generalizes perfectly to the four tests (bottom).

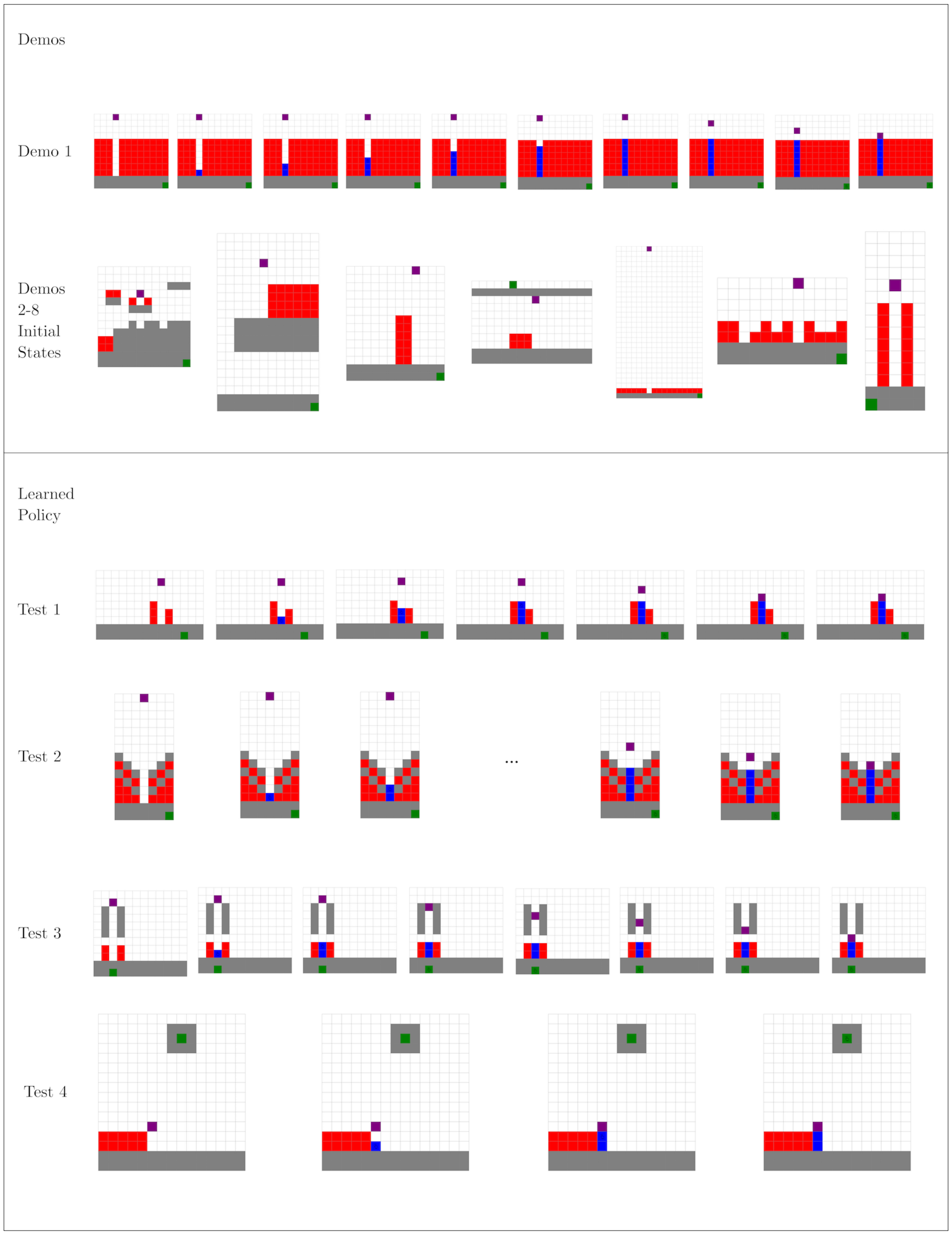


Figure 9: Demonstrations and generalization tests for the "Chase" game. After eight demonstrations (top), the learned policy generalizes to win all four test games (bottom). Note that one more blue block is drawn than necessary in Test 2, suggesting that the learned policy tries to avoid contact between the purple falling object and any object. This is more conservative play than required by the game, but the demonstrations evidently are not sufficient to rule out the possibility that gray blocks should be avoided.

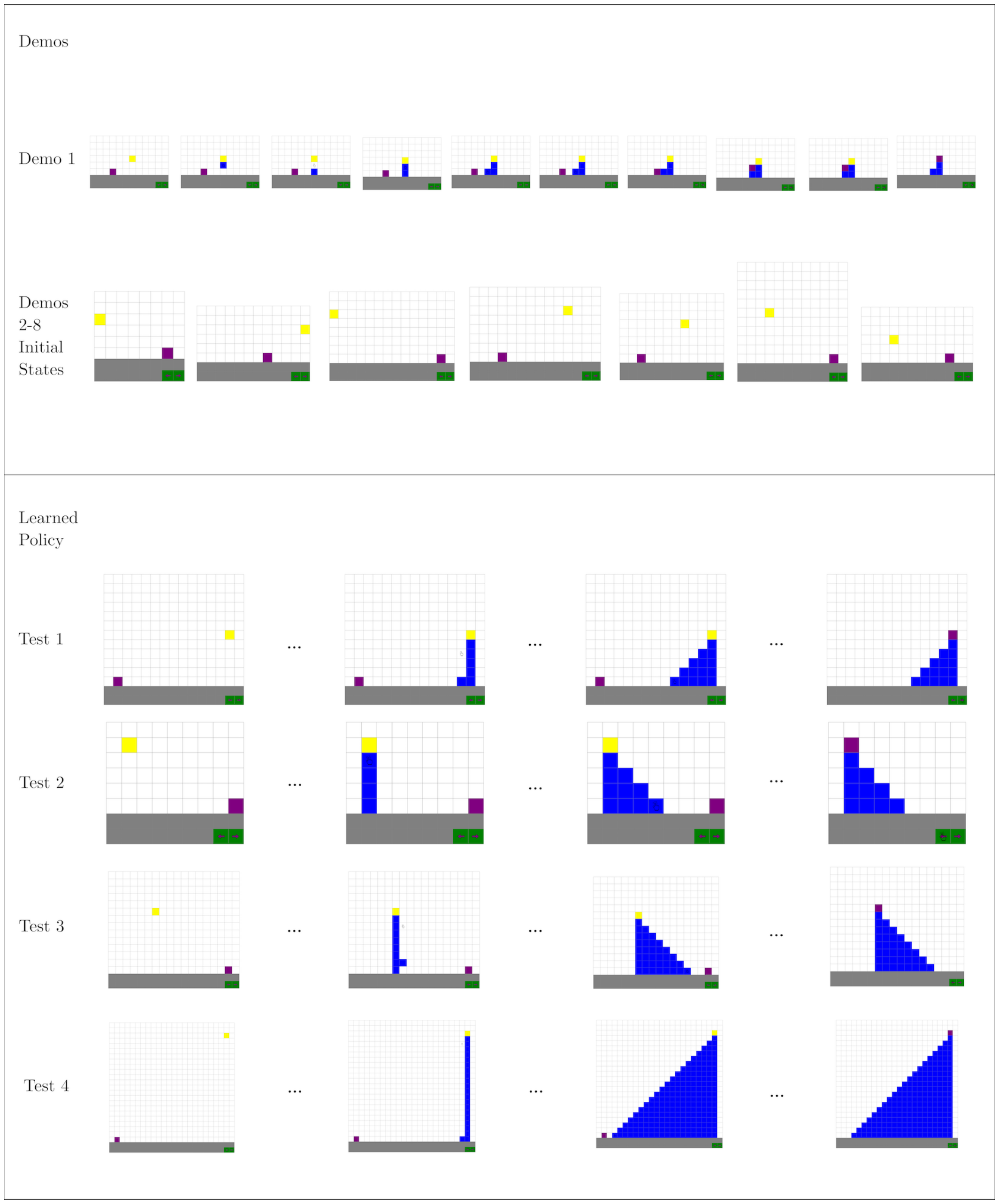


Figure 10: Demonstrations and generalization tests for the "Reach for the Star" game. After eight demonstrations (top), the learned policy generalizes perfectly to the four tests (bottom), including tests with much larger grids and higher stars than seen in the demonstrations.