MINIMAL COMPLETE SPECIFICATION LANGUAGES VIA SPECIFICATION EQUIVALENCE BACKUPS

Anonymous authors
Paper under double-blind review

ABSTRACT

As AI systems become increasingly able to solve any task we can specify, the bottleneck becomes being able to more efficiently and robustly specify our tasks. In this work, we move towards the theoretical limit of optimal specification: finding the minimum specification language which allows for precise specification of any task in a domain. We formalize the Specification Language Design Problem, and define the minimum amount of information necessary to determine an optimal policy for the specification, which we call the Specification Equivalence Set. This set has the recursive substructure necessary for a Bellman backup to compute it exactly, allowing us to find optimal specification language efficiently and exactly for a set of specification problems. We then extend these insights to a deep RL algorithm, using a VQVAE-inspired method to train an agent that is controllable via a latent specification language. We find that this deep RL algorithm matches our exact algorithm at small scales, and allows the system to scale to larger domains. In introducing this problem and our methods of addressing it, we aim to provide a primary look into the benefits of more precise specifications for AI systems.

1 INTRODUCTION

We are increasingly used to controlling AI systems by directly telling them what we want via natural language prompting. However, when using RL algorithms in systems we do not already understand, we may not yet know what we need to specify to communicate our intent, and we may not know how to put our intent into words (Kwon et al., 2023). As AI systems push the frontier of human knowledge, it will more often be the case that we do not yet know what to ask for, how to ask, or how to guarantee it is done safely.

It may help in these cases for the AI system to provide a compressed space of objectives for us. There is a significant body of work in RL working towards world models, latent representations of states, and goals which compress an agent's understanding of the problem, leaving only what is essential (Schaul et al., 2015; Barreto et al., 2020; Lamb et al., 2022; Hafner et al., 2025). These approaches already represent a significant push towards a compressed representation, which may be useful for understanding and specifying human intent.

However, these representations contain much more information than is relevant to the task of specifying human intent, especially as we plan for a future where agents are generally capable at solving tasks that their human users cannot solve. A human directing the RL agent does not need to know how to solve the task themselves, they simply need a sufficient degree of control to effectively express their intentions.

Similar to Sutton's bitter lesson (Sutton, 2019), it seems likely that the human effort necessary to correctly express their intent could increasingly become the bottleneck in using these systems effectively. To this end, we aim to move towards the theoretical limit of specification. That is, we want to understand how little human effort a system could require while still allowing the human to exactly specify their intent. Importantly, given human effort is the hypothesized bottleneck, we will accept trading compute to drive the human effort necessary to accurately specify intent to a minimum, assuming the RL algorithm has sufficient data and compute from the environment.

To that end, we define the *Specification Language Design Problem* as the problem of creating a set of choices that a user must make to define their task. We have two criteria for solving this problem.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

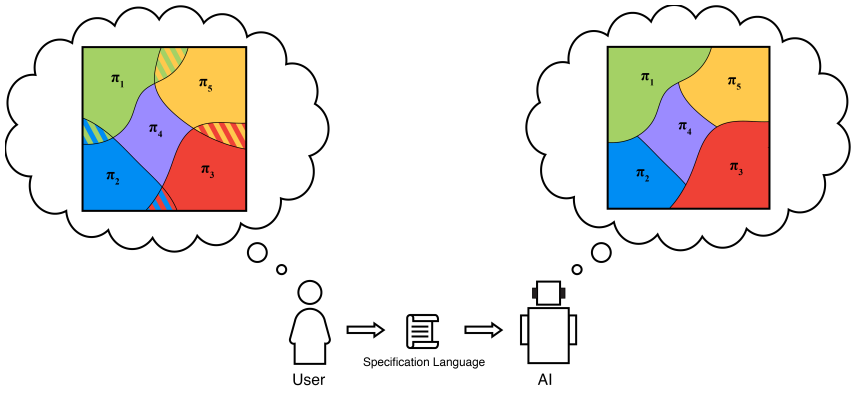

Figure 1: An Abstract representation of a Specification Language, in which the square represents some UMDP space of tasks with optimal policies show in color. The transformation is shown from UMDP to Specification Language, where the policy space is compressed. Rather than needing to specify some exact point in the space of task, the user can instead use the simplified language and specify only up to the single colored region which holds the intended task with no need to specify any further.

First we want the specification language to be *complete*, in that any task the human might consider specifying can be expressed in this language. Second, under the constraint of completeness we want the specification language to be *minimal* in that it requires the minimum amount of human input.

We prove that under these two conditions, the unique specification language that minimizes human input corresponds to the minimum cover of tasks by the *maximal Specification Equivalence Sets*. Moreover, building on the work of Lauffer et al. (2023), we provide an exact Bellman backup algorithm Specification Equivalence Backup (SEB) which can efficiently compute this object exactly. We show the structure of these specification languages in a series of grid worlds. Finally, to move towards a scalable approach to approximating Specification Equivalence Sets, we propose a Deep-RL algorithm, Approximate Learning of Specification Equivalence Sets (ALSES), which aims to approximate these specification languages.

2 Related Work

Reasoning about Specifications: In the work of Hadfield-Menell et al. (2017), they explore the idea that the given reward function might be a fallible proxy for the true designer intent and instead learn to group specifications of the reward function by the behavior they induce. Sorg et al. (2010) follows the same intuition about the reward function specification. They introduce a designer level reward and, through gradient ascent, optimize the agent reward to maximize the designer reward. Similarly, Bayesian active reward learning maximally reduces human-reward-choices posterior volume (Sadigh et al., 2017), and Information-directed reward learning (Lindner et al., 2021) instead targets uncertainty about the difference in return between plausibly optimal policies.

Compressing populations in Multi agent RL: A significant basis for this work is from a related question in multi-agent RL. Specifically, Lauffer et al. (2023) aims to determine the minimum information one needs about a rational cooperator's policy to know a best response. They introduce the concept of Valued Strategic Equivalence Relations, an equivalence relation on the space of co-policies where they are equal iff they induce the same expected return and Best Response set by the player. They layout a backup-Bellman style, backwards induction algorithm that takes in a Decentralized Partially Observable MDP and results in a Directed Acyclic Graph (DAG) compressing the necessary information about the co-policies to respond optimally. Our Specification Equivalence Backup (SEB) is a related-family algorithm of this construction where we replace minimal information for cooperation with minimal specification. Further work in Deep RL methods like in Rahman et al. (2024) generate a teammate population who approximate best responses for any partner.

Latent Representation and goals: A large body of RL research compresses the agent's view of the problem into a latent state. World models (Hafner et al., 2025) and state-representation learning

seek low-dimensional states that retain what matters for control (Lesort et al., 2018) or dynamics. Discrete codebooks (Van Den Oord et al., 2017) give such latents a finite vocabulary, which ALSES exploits. A parallel line compresses the space of tasks rather than the space of states. Universal value function approximators condition values on a goal embedding (Schaul et al., 2015). These representations are compressed for the agent's purposes retaining whatever is needed to *solve* the task. Our minimality criterion is stricter, since a representation for *specifying* a task need only distinguish tasks that require different behavior.

Human–computer and human–robot interaction: In shared autonomy a person supplies low-dimensional, noisy input and the system infers the goal and completes the task; Reddy et al. (2018) learn this mapping end to end with model-free deep RL, using task reward as the only supervision and a single parameter to set how much control the user retains. Learned latent actions design the channel itself, embedding a robot's actions into a few dimensions a person can drive with a joystick (Losey et al., 2020; 2022). These works share our premise that the person need only steer the task, not solve it. We design our specification languages around the minimal size of the language to represent what the channel must carry.

3 Background

Underspecified MDPs (UMDPs). In order to define the problem of task specification, we need a space of possible tasks. Here we model this space using Underspecified MDPs, defined as a tuple $\langle S, A, \Theta, T, R, \gamma \rangle$, where S is a set of states, A is a set of actions, Θ is the free parameters of the environment, $T : S \times A \times \Theta \to S$ is a transition function across all parameterizations of the UMDP, $R : S \times \Theta \to \mathbb{R}$ is the reward function of a state, and $\gamma \in [0, 1]$ is the discount factor. At every timestep, the agent knows its state $s_t \in S$, chooses an action $a_t \in A$, transitions to its new state from the transition function $s_{t+1} = T(s_t, a_t, \theta)$ where $\theta \in \Theta$ represents the current parameterization, and receives the reward of the new state from $r_{t+1} = R(s_{t+1}, \theta)$. We mark $\pi \in \Pi$ as a policy in this UMDP, a decision rule for choosing actions in each state, and define the value of a policy π as $U^w(\pi) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$ and $U^w(\pi^*) = \arg\max_{\pi \in \Pi} \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$. More concretely, the free parameters θ represent different settings of the possible UMDP such as the existence and position of hazards in a maze. Continuing with terminology from related works, we refer to a single setting θ as a *level*.

4 Specification Language Design Problem

In the Specification Language Design Problem, given a set of possible specifications, we aim to design a specification language which minimizes the number of choices the designer must make while still resulting in the optimal policy for the designer's true task. That is, the specification language design problem removes any redundant information that a policy does not need to remain optimal, leaving only the essential information. Given a UMDP, the aim of the Specification Language Design Problem is to design a specification language $\langle \Sigma, \Lambda \rangle$ where Σ is a set of possible explicit specifications and $\Lambda : \Sigma \to \Pi$ is an interpretation of those specifications to result in a policy.

In this paper, we will be concerned with exact solutions to the specification language design problem which are optimal for designers who know how to use them optimally. Thus, we want to ensure that for any complete task that the designer could have in mind $w^T \in \Theta$ there exists an explicit specification which would result in the optimal policy on that task. More formally, $\exists \sigma \in \Sigma$ s.t. $\Lambda(\sigma) \in \arg\max_{\pi \in \Pi} \{U(\pi, w^T)\}$. Simultaneously, we want this specification language to require as few choices from the AI designer as possible, so $|\Sigma|$ is minimized. Combined, these two objectives together define a Minimal Complete Specification Language (MCSL). We consider this a 1-shot interaction, where the designer cannot supply more input throughout the episode.

5 Computing Minimal Complete Specification Languages via Specification Equivalence Backup - SEB

In this section, we introduce this work's main contribution: an exact algorithmic answer to the Specification Language Design Problem when the number of levels is enumerable.

It's useful to understand the structure of Minimal Complete Specification Languages in order to better motivate our algorithm. A designer with any intended task $w^T \in \Theta$ for some given MCSL $\langle \Sigma, \Lambda \rangle$ and UMDP $M \sim \langle S, A, \Theta, T, R, \gamma \rangle$ would not need to specify every aspect of the task. Instead, it would be sufficient to specify only up to a clearly optimal policy. That is, they would be satisfied with any specification $\sigma \in \Sigma$ which results in an optimal policy $\pi^\sigma = \Lambda(\sigma)$ for σ. This means that if there are many tasks which share an optimal policy, they can share a specification. This idea is demonstrated in Figure 1. For example, if a single policy is optimal in every parameterization, no specification is required, and the agent can instead play the same policy in all levels. Conversely, if all levels require unique policies, then the specification must specify down to the exact level.

In every other case, we must partition the parameterizations into different sets which share an optimal policy. Each specification σ thus allows the designer to exactly express their task only on the set of worlds where $\Lambda(\sigma)$ is optimal. We call such sets *Specification Equivalence Sets* and define *Maximal Specification Equivalence Sets* as Specification Equivalence Sets which cannot be made larger while remaining Specification Equivalence Set. Note that in general maximal Specification Equivalence Sets generally overlap, since levels may share an optimal policy with several sets that share none with each other.

Definition 1 (Specification Equivalence Set)**.** A set $W \subseteq \Theta$ is a Specification Equivalence Set iff there is some policy optimal in every member: $\exists \pi \in \Pi \, \forall w \in W : \; V^w(\pi) = V^w(\pi^*_w)$. $\mathcal{S}(M)$ denotes the collection of all Specification Equivalence Sets of M.

Definition 2 (Maximal Specification Equivalence Set)**.** A Specification Equivalence Set W of M is *maximal* iff no strict superset of W is a Specification Equivalence Set: for every $w \in \Theta \setminus W$, $W \cup \{w\} \notin \mathcal{S}(M)$. $\widehat{\mathcal{S}}(M) \subseteq \mathcal{S}(M)$ denotes the set of maximal Specification Equivalence Set. Because subsets of Specification Equivalence Sets are Specification Equivalence Sets, $\widehat{\mathcal{S}}(M)$ determines $\mathcal{S}(M)$: $W \in \mathcal{S}(M)$ iff $W \subseteq N$ for some $N \in \widehat{\mathcal{S}}(M)$.

Given these definitions, if $\langle \Sigma, \Lambda \rangle$ is a MCSL then we can identify each σ with the Specification Equivalence Setfor which $\Lambda(\sigma)$ is optimal. Moreover, without loss of generality, these sets should be maximal, as this would strictly improve the versatility of the specifications. Finally, because the language is *Complete*, every level must have an optimal policy represented in $\Lambda(\sigma)$ for some sigma, so every level is in one of these Specification Equivalence Sets. Putting this together, to minimize designer input while maintaining completeness, every task must map to a single policy with the corresponding Specification Equivalence Sets, and it should be a minimal cover of Θ, one of smallest cardinality. We show such a minimum cover of Θ and a MCSL are the same object.

Theorem 1 (MCSLs correspond to minimum covers by the maximal Specification Equivalence Sets)**.** *Let Σ be a partition of Θ and let $\Lambda : \Sigma \to \Pi$ be such that $\Lambda(\sigma)$ is optimal in every $w \in \sigma$ for each $\sigma \in \Sigma$. Then $\langle \Sigma, \Lambda \rangle$ is a Minimal Complete Specification Language if and only if Σ is a minimum cover of Θ by $\hat{\mathcal{S}}(M)$.*

Proof. Let k be the cardinality of a minimum cover of Θ by $\hat{\mathcal{S}}(M)$ and m the least number of specifications in any Complete language. $\langle \Sigma, \Lambda \rangle$ is Complete, since every w lies in some σ and $\Lambda(\sigma)$ is optimal in w. Therefore, it is an MCSL iff $|\Sigma| = m$, and it is a minimum cover iff $|\Sigma| = k$, meaning it suffices to show that $m = k$.

$k \leq m$: Let $\langle \Sigma', \Lambda' \rangle$ be Complete. For $\sigma' \in \Sigma'$ let $W_{\sigma'}$ be the set of parameterizations in which $\Lambda(\sigma')$ is optimal. By the finiteness of Θ and the fact that it is a Specification Equivalence Set, it must lie within some $N_{\sigma'} \in \hat{\mathcal{S}}(M)$. By the completeness, $W_{\sigma'}$ with all σ' cover Θ and hence so do $N_{\sigma'}$ with all σ'. There are at most $|\Sigma'|$ of them so $k \leq |\Sigma'|$ since a cover created from a specification has no more sets than the language had specifications.

$m \leq k$: Let Σ^* be a minimum cover of Θ by $\hat{\mathcal{S}}(M)$ and let $\Lambda^*(\sigma^*)$ be optimal in every member of σ^*, which exists since σ^* is a Specification Equivalence Set. Then $\langle \Sigma^*, \Lambda^* \rangle$ is Complete with $m \leq |\Sigma^*| = k$.

Hence $m = k$. If Σ is a minimum cover then $|\Sigma| = k = m$, so $\langle \Sigma, \Lambda \rangle$ is an MCSL. If $\langle \Sigma, \Lambda \rangle$ is an MCSL then $|\Sigma| = m = k$, so Σ is a minimum cover. □

The algorithms detailed in the next subsection takes as input a UMDP M, and output a Directed Acylic Graph (DAG) whose root nodes are exactly the $\hat{\mathcal{S}}(M)$. This resulting DAG is used to find the

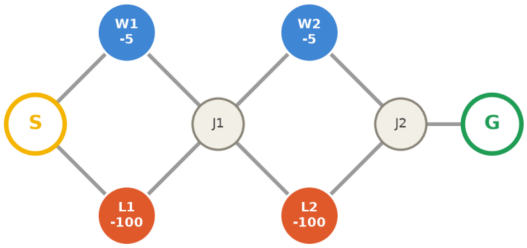

Figure 2: A toy UMDP of the graph above, with 7 possible levels. They are $\{\text{safe}, w1, w2, w1l1, l2, w2, w2l2\}$ where each of those labels represent a world where the hazard exists, meaning the agent will incur the penalty listed, or is missing, meaning the agent gets no penalty for walking through it. The agent only learns if the hazard exists after entering the node.

$\Lambda(\sigma)$ for each Specification Equivalence Set node in the root by following the nodes labeled with the chosen specification down the graph. To get the MCSL answer to the Specification Language Design Problem we then run a minimum cover algorithm on those roots to obtain our MCSL.

5.1 Algorithmically Finding MCSLs

With the theoretical groundwork laid out we introduce the Specification Equivalence Set Backup (SEB) algorithm. Given a UMDP, the SEB algorithm runs a backup-Bellman style backwards induction on the states within each level while combining nodes across the entire UMDP. The result is a DAG where each node represents the maximal Specification Equivalence Sets of that state and time. To combine and remove redundant specification, within each node each action holds the worlds in which it is optimal, creating one step Specification Equivalence Sets. These best response sets (BR sets) are found at each timestep by running a standard Bellman-backup on each level, prior or concurrently to this algorithm. Extending these sets past one step optimality, each are intersected with the sets in the following child nodes to create the Specification Equivalence Sets for the subsequent subgame. After removing empty sets and subsets of other intersections, the result is the maximal Specification Equivalence Sets for that state and time with the algorithm continuing by inducting backwards. A proof that SEB finds the maximal Specification Equivalence Sets at every state and time is in Appendix A.2.

We can demonstrate this entire algorithm concretely with the UMDP found in Figure 2. We enumerate the 7 possible settings of this UMDP as $\{\text{safe}, w1, w2, l1, l2, w1l1, w2l2\}$ where world $w1$ has only the first water tile hazard while world $w1l2$ has the first water hazard and the second lava hazard, etc. The agent never knows if a node is safe or harmful before stepping in it.

Walking through the algorithm, we would start at the last state G where everything is optimal, saving every world label into the same node $\{\text{safe}, w1, w2, w1l1, l2, w2, w2l2\}$. This repeats, similarly to a bellman backup, until the timestep reaching $J1$ where the agent has its first choice, "up" or "down". With two choices, we partition the levels by the actions that are optimal resulting in up $\rightarrow \{\text{safe}, l1, w1, w1l1, l2, w2l2\}_{\text{up}}$ and down $\rightarrow \{\text{safe}, l1, w1, w1l1, w2\}_{\text{down}}$.

For both of those nodes, we intersect it with the child nodes of the resulting state. Here shown just for the "up" action $\{\text{safe}, l1, w1, w1l1, l2, w2l2\}_{\text{up}} \cap \{\text{safe}, w1, w2, w1l1, l2, w2, w2l2\}_{J2}$. At every node, after the intersection, we remove sets which are subsets of others along with empty sets. Continuing to the start node we would end with 4 Specification Equivalence Set:

$$\{\text{safe}, l1, w1l1, l2, w2l2\}_{\text{up,up}} \qquad \{\text{safe}, l1, w1l1, w2\}_{\text{up,down}}$$
$$\{\text{safe}, w1, l2, w2l2\}_{\text{down,up}} \qquad \{\text{safe}, w1, l1\}_{\text{down,down}}$$

Importantly, this is not the final output of the algorithm. These are instead the maximal Specification Equivalence Sets, the final step is to run a minimum cover algorithm on this $\widehat{\mathcal{S}}(M)$ and get our final partition:

$$\{w1, w2\}_{\text{up,up}} \qquad \{\text{safe}, l1, w1l1, l2, w2l2\}_{\text{down,down}}$$

As a note, the "safe" world could have gone in either set since both are optimal, though the resulting MCSL is still provably minimal. Additionally, this is not the entire algorithmic representation of

the node, the node object itself in the code, both points to child nodes for continuations, and holds its set of optimal actions. We can then use this maximal $\mathcal{S}(M)$ DAG to play optimally in any level by sampling from actions' sets that hold that level's parameterization. The pseudocode for SEB can be found in Appendix A.1 in both a value-less version and a valued version (more details about the valued form are in Appendix A.3).

6 Deep RL approximation - Approximate Learning of Specification Equivalence Sets (ALSES)

Although the algorithm detailed above is exact, it does not scale well to high-dimensional or combinatorial environments. In this section, we layout a preliminary method for a scalable solution.

To make the solution scalable, we use samples of levels from the UMDP as input, and elicit a model to partition the worlds into classes based on their optimal policies. This approach is inspired by VQVAEs (Van Den Oord et al., 2017), where an encoder head encodes the levels into embeddings which get snapped to the nearest code and passed as input into a policy net. Concretely, levels with the same optimal policy would get embedded to the same code, the number of learned codes being a hyperparameter of the model. The policy network, alongside the code, gets state based information such as coordinate position in mazes to choose actions in each state, but is *not* given enough information about the full level in order to play optimally without the code.

The training involves a level input, the encoder then sends it to a code, and the policy network uses the code to play. The whole network is trained with REINFORCE with critic baseline. Similarly to a VQVAE, the encoder is additionally has a commitment and codebook loss terms, but with the policy gradient term replacing the reconstruction loss. Recovering the true partition while maintaining a minimal codebook is the global minimum and forces the encoder to encode the levels together by optimal policy. To encourage codebook minimality, we add a loss term as the entropy of code usage:

$$\mathcal{L}_{\text{use}} = -\lambda(t) \sum_{k=1}^{K} \bar{p}_k \log \bar{p}_k, \quad \bar{p}_k = \tfrac{1}{N} \textstyle\sum_i \operatorname{softmax}_k\big(- \|z_i - c_k\|^2 / \tau \big).$$

We train in different phases. First, there are warmup phases where we set the penalty λ above to 0 or even negative to encourage code use and where we restart unused codes on top of current embeddings of levels chosen randomly. Next, there are penalty phases where codes are not restarted and the penalty term is > 0. Additionally, the training of the critic network gets to see the exact level rather than the learned encoder code to stabilize training. It uses standard policy-entropy bonus in order to keep exploration levels high since routing all policy through codes can lead to quick behavior collapse. The policy-entropy bonus used is shown below.

$$\mathcal{H}_\pi = -\frac{\beta}{NL} \sum_{i=1}^{N} \sum_{t=1}^{L} - \sum_{a \in A} \pi(a \mid s_{i,t}, k_i) \log \pi(a \mid s_{i,t}, k_i)$$

7 Evaluations

For all evaluations in this work we use 2d gridworlds across both SEB and ALSES methods. Every experiment uses a UMDP represented as possible levels. If one were to take the perspective of the designer, this UMDP would act as the underspecified training worlds for the agent, meaning that by finding the specification language from our algorithms, the designer would be able to narrow down towards the policy desired. Reoccurring tiles in these evaluations will be wall tiles, always colored in black to form maze like environments, and differently colored hazard tiles (like red for lava, or blue for water). At no point in any of these experiments are the tiles revealed to be harmful or not to the agent. We show both methods supply the desired MCSL for each UMDP.

7.1 Four-World Maze

We first demonstrate SEB in an intuitive UMDP we call the four-world maze. In this setting, we have 4 possible parameterizations, that can turn on or off a lava or water tile. This is the problem

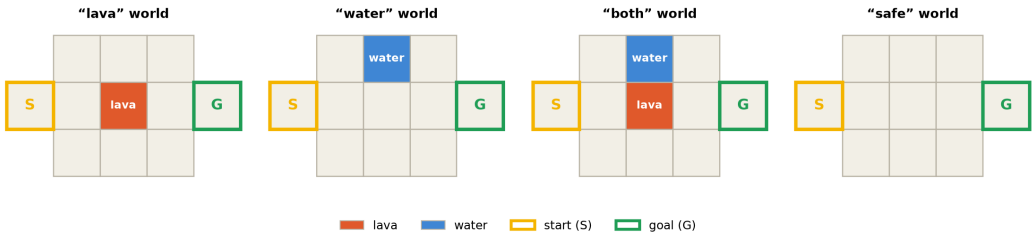

Figure 3: The four mazes represent a single UMDP with 4 parameterizations. The agent can move in any adjoining empty cell, and stepping on an existing hazard will always give a penalty of -100.

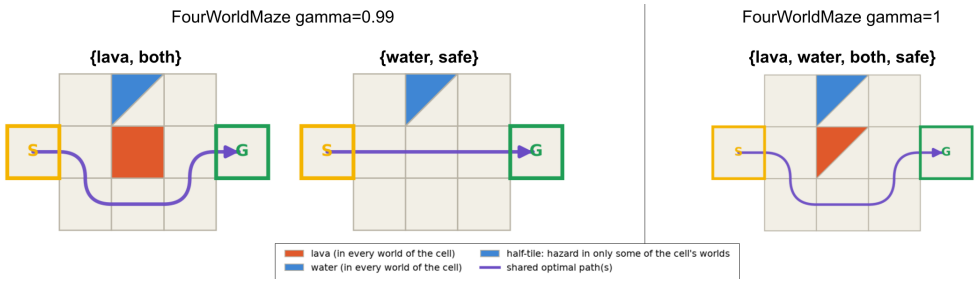

Figure 4: Above are the recovered specifications from running SEB on the UMDP with $\gamma = 0.99$ and $\gamma = 1$. When the length of the path matters, the agent needs a specification in order to find the optimal policy, while in the case where the optimal solution does not care about path length, the agent does not need any specification, opting to always take the bottom path.

represented in Figure 3. With a discount $\gamma = 0.99$, the optimal path is then intuitively through the middle when there is no lava, and around it with lava.

SEB does indeed recover the desired optimal partition where the lava and both levels combine and water and safe combine. From this we create our MCSL, which can be narrowed down a single question: "Is lava there or not?". After answering this question, the agent would know exactly how to act within its environment. Interestingly, running the same experiment with $\gamma = 1$, there is no need for the shortest path, and the bottom path becomes optimal in every level, meaning SEB reduces the problem to a single Specification Equivalence Set as expected. Both of these results are shown in Figure 4. This can be interpreted as the model only needing specification when it is necessary to the optimal policy, ie: *the agent always finds the policies which require the least specification*.

7.2 ALSES ON A NOISY UMDP

Using the same UMDP as 7.1, we demonstrate the necessity of ALSES in a setting in which the number of worlds is increased by injecting random noise into each level to create new parameterizations. Specifically, we describe each parameterization as one of the four possible mazes, but include a 32bit integer that modifies the reward function. We take the first 8 bits of the 32bit integer, and use it as increments of 5 for the penalty for stepping on a hazard, giving an effective hazard penalty range between -5 at the minimum and -300 at the maximum. The SEB algorithm would need to enumerate and solve every level individually. We train the ALSES algorithm starting with 4 learned codes, by sampling random worlds and feeding the 32bit integer as input into the encoder.

Even with this, on eight seeds we find that ALSES discovers the true Specification Equivalence Sets 50% of the time, finds a slightly suboptimal partitioning where all levels take the bottom path only 25% of the time, in seed 2 a majority but not all seeds get partitioned in their true sets, and gets stuck learning in seed 6. These results are showing in Figure 9 in Appendix A.5. The method is able to learn the right MCSL a majority of the time even with noise added to scale problems.

As another experimental set up, the noise stays part of the parameterization fed into the encoder, but does not affect the reward function. The results for this setup are also shown in the Appendix A.5 in Figure 10. The trend is the same, recovering the true partition around 50% of the time, and recovering exact behavior, though imperfect codes another 25%.

7.3 COLORED KEY MAZE

Scaling up the experiments, we now increase both the size and the complexity. In this UMDP, which we refer to as the "colored key maze," we introduce colored tiles which are possibly trapped passageways. The agent can explore to find a colored key to disarm the corresponding colored passageway regardless of if they are trapped or not, meaning no penalty will be taken no matter the current parameterization. In this example, we set the penalty of the $pink$ and $blue$ passageway without the key in the parameterization in which they are trapped to -50 while setting the penalty of going through a red passageway without a red key in a level where they are trapped to only -4. This will cause the agent to willingly take that penalty in their optimal policy as the detour would cost more. This UMDP can be seen in Figure 5.

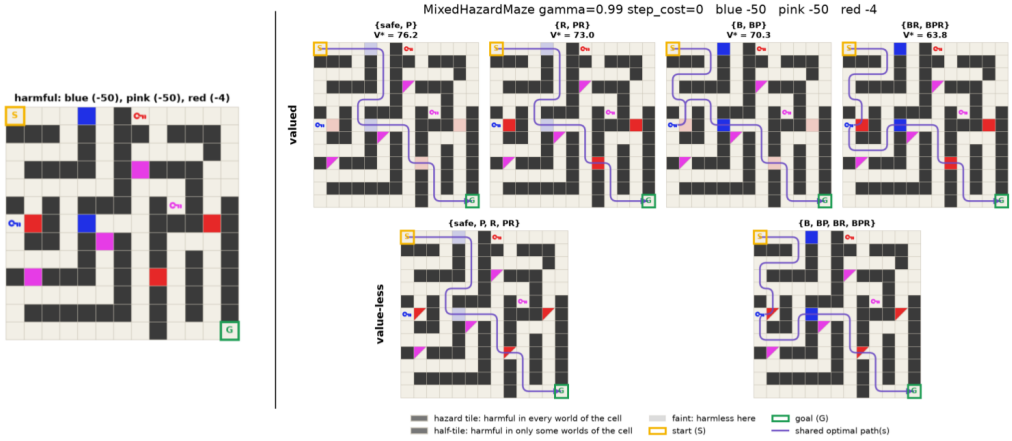

Figure 5: The left figure displays the UMDP with 8 possible worlds as the binary combinations of each color passageway being trapped or not. On the right, the partitions are the results of running the SEB algorithm, both when you partition on both policy and final value and when you partition only on policy, and show the optimal policy for each Specification Equivalence Set. The labels above each maze show which worlds belong in that set.

We take this UMDP and solve it with SEB in its value-less and valued version (Appendix A.3), and are able to see the actual difference in partitions in Figure 5. The valued version of the algorithm cares about the return, so we see that the partitions are split, not only by policy, but by expected value too. In contrast, the value-less version foregoes the value and partitions purely on optimal policy. From the value-less version, we learn that the only question required for specification in this UMDP is whether blue is harmful or not. As such, we learn that any question or information about pink or red is entirely unnecessary to play optimally in the value-less setting.

7.4 ALSES COLORED KEY MAZE

Applying the same UMDP as the above in section 7.3 to ALSES, we set the number of codes to 8 and run 8 seeds of training runs with a gradual code entropy penalty. We begin with no penalty and grow λ from $0 \rightarrow 0.1$ between training update 500 and 1000. The results of this experiment are in Figure 6. The ALSES algorithm recovers the optimal partition in 25% of the seeds, finding both the right policy and Specification Equivalence Sets. We note that if a policy is not learned or not common in that code before codes merge, it will be entirely forgotten. This suggests that the no penalty phase, used to learn to play the worlds optimally before adding code usage penalties, is highly beneficial.

7.5 ALSES LAVA MAZE

We further scale up the experiments using a UMDP with many more parameterizations in the same grid layout, but remove the colored keys and colored doors. Instead, we layout possible lava tiles in 11 positions around the maze, giving us a combinatorial 2048 possible parameterizations. We run

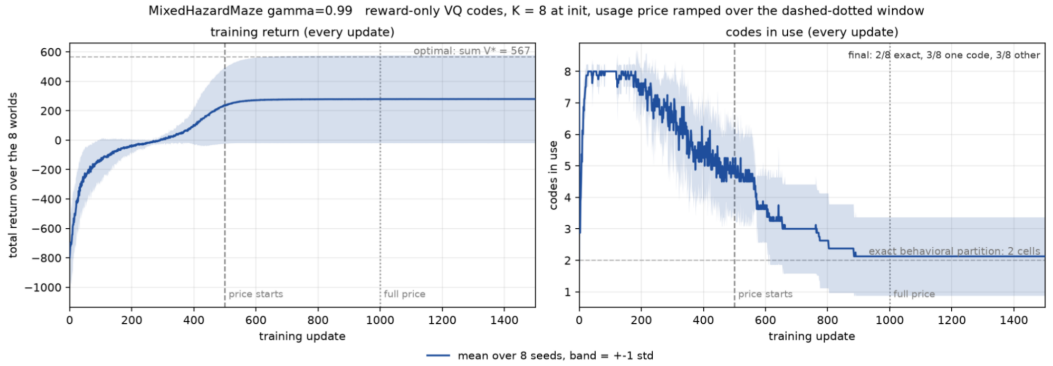

Figure 6: The left figure shows the average learned reward across the 8 seeds for the ALSES algorithm, in which it learns a specification language for the mixed hazard maze in the figure above. The vertical dashed lines shows the start and end of the schedule for the λ penalty on the code usage. The right figure shows the average codes in use.

both SEB and ALSES on this UMDP to show the true partition. We demonstrate and analyze the learned results in Figure 7 in the Appendix A.4.

Overall, these ALSES experiments show a promising direction with partition recovery for specification. However, this method could be improved in its current form to increase the chance to recover the exact MCSl. We anticipate that since it relies on having found the policy before the merge and keeping newly found routes rather than outvoting them in each code, better merging objectives, or exploration could have a large positive impact on the stability of this algorithm.

8 Discussion

This work introduced the Specification Language Design Problem: Given a set of possible specifications, we must come up with a specification language which allows a designer to specify their desired optimal policy. We then detail how to answer this problem algorithmically on small examples and prove that a backup-bellman inspired Specification Equivalence Backup (SEB) algorithm gives us a Minimal Complete Design Language. We laid out Approximate Learning of Specification Equivalence Sets (ALSES), a scaleable, deep RL method and demonstrated it experimentally recovering the desired MCSL in mean performance on toy UMDPs.

Future works would naturally extend these methods. SEB modified to fit partially-observable settings or stochastic transitions would introduce a new dimension to the specification problem, one in which information about the environment is revealed to the agent over time from these non-certain continuations, and in which an agent could then ask questions temporally throughout, only when faced with a situation that necessarily demands it. Concretely, if the first steps in a maze are certain and later diverge stochastically, then only after the first divergence does the agent need a specification about its current path. Secondly, future work into the instability of the ALSES method would help increase the reliability and scale of the method, allowing for specification languages to be used in much larger, real-world scenarios. We anticipate that stability improvements would come from a different exploration and policy entropy mechanism to allow the model to find and adopt more intricate routes. and from different objectives for code merges.

Finally, in terms of limitations, in this work we make the assumption that both the designer and the agent are able to use the specification language optimally, such that given a specification language, the designer knows the right choices to specify the true objective and the agent uses the specification language to recover the desired policy. In real use cases, it is reasonable to assume that one or both of these conditions would not hold and would require further refining of the problem.

All in all, we are excited about the introduction of Specification Language Design Problem and the solutions we demonstrate in this paper. As models begin solving problems above human understanding, accurate specification of problems will increasingly becomes important for safe and capable systems.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for helping write proofs and helping implement methods. We have not used generative AI tools for developing theoretical models or conceptual frameworks, formulating mathematical claims, supplying key ingredients for proofs, proposing or refining hypotheses, designing or giving feedback on methodology/experiments, qualitative/thematic data analysis, interpreting results, and generating synthetic datasets, translation, cleaning or reformatting datasets, are not applicable to this work. Additionally, we used generative AI tools for finding relevant literature, general information search, writing or editing code for figures, formatting references. We have reviewed all AI-assisted work. All code was double checked by the main author for completeness and accuracy, and all literature review were likewise checked for correctness and fit. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

See the ICLR 2027 AI Policy for Authors for more details. This statement should not be more than 1 page.

ETHICS STATEMENT

(This section is **recommended** and does not count toward the page limit.)

If authors feel that their paper submission raises questions regarding the Code of Ethics, they are encouraged to include a paragraph of Ethics Statement (at the end of the main text before references) to address potential concerns where appropriate. Topics include, but are not limited to, studies that involve human subjects, practices to data set releases, potentially harmful insights, methodologies and applications, potential conflicts of interest and sponsorship, discrimination/bias/fairness concerns, privacy and security issues, legal compliance, and research integrity issues (e.g., IRB, documentation, research ethics). This statement should not be more than 1 page.

REPRODUCIBILITY STATEMENT

(This section is **recommended** and does not count toward the page limit.)

It is important that the work published in ICLR is reproducible. Authors are strongly encouraged to include a paragraph-long Reproducibility Statement at the end of the main text (before references) to discuss the efforts that have been made to ensure reproducibility. This paragraph should not itself describe details needed for reproducing the results, but rather reference the parts of the main paper, appendix, and supplemental materials that will help with reproducibility. For example, for novel models or algorithms, a link to an anonymous downloadable source code can be submitted as supplementary materials; for theoretical results, clear explanations of any assumptions and a complete proof of the claims can be included in the appendix; for any datasets used in the experiments, a complete description of the data processing steps can be provided in the supplementary materials. Each of the above are examples of things that can be referenced in the reproducibility statement.

REFERENCES

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117 (48):30079–30087, 2020.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. *Advances in neural information processing systems*, 30, 2017.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.

Alex Lamb, Riashat Islam, Yonathan Efroni, Aniket Didolkar, Dipendra Misra, Dylan Foster, Lekan Molu, Rajan Chari, Akshay Krishnamurthy, and John Langford. Guaranteed discovery of control-endogenous latent states with multi-step inverse models. *arXiv preprint arXiv:2207.08229*, 2022.

Niklas Lauffer, Ameesh Shah, Micah Carroll, Michael D Dennis, and Stuart Russell. Who needs to know? minimal knowledge for optimal coordination. In *International Conference on Machine Learning*, pp. 18599–18613. PMLR, 2023.

Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

David Lindner, Matteo Turchetta, Sebastian Tschiatschek, Kamil Ciosek, and Andreas Krause. Information directed reward learning for reinforcement learning. *Advances in Neural Information Processing Systems*, 34:3850–3862, 2021.

Dylan P Losey, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. Controlling assistive robots with learned latent actions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 378–384. IEEE, 2020.

Dylan P Losey, Hong Jun Jeon, Mengxi Li, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, Jeannette Bohg, and Dorsa Sadigh. Learning latent actions to control assistive robots. *Autonomous robots*, 46(1):115–147, 2022.

Muhammad Rahman, Jiaxun Cui, and Peter Stone. Minimum coverage sets for training robust ad hoc teamwork agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17523–17530, 2024.

Siddharth Reddy, Anca D Dragan, and Sergey Levine. Shared autonomy via deep reinforcement learning. *arXiv preprint arXiv:1802.01744*, 2018.

Dorsa Sadigh, Anca Dragan, Shankar Sastry, and Sanjit Seshia. Active preference-based learning of reward functions. 2017.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

Jonathan Sorg, Richard L Lewis, and Satinder Singh. Reward design via online gradient ascent. *Advances in Neural Information Processing Systems*, 23, 2010.

Richard S. Sutton. The bitter lesson. Blog post, Incomplete Ideas, March 2019. URL `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

A Appendix

A.1 Pseudocode for SEB

Algorithm 1 Backup of maximal specification-equivalent sets (valued or value-less)

Require: UMDP $M = \langle S, A, \Theta, T, R, \gamma \rangle$; optimal-action sets $BR_w(s,t)$ and values $V^*_w(s,t)$ for every $w \in \Theta$; flag VALUED
Ensure: $\mathcal{F}(s,t)$, the maximal (valued) Specification Equivalence Set at every reachable node, each with a record (a, N); the language $\langle \Sigma, \Lambda \rangle$ at the root

$\mathcal{F}(s,H) \leftarrow \{\Theta\}$ for every s
for $t = H-1$ **down to** 0 **do**
  for each state s reachable at time t **do**
    if VALUED **then**
      $\mathcal{C} \leftarrow$ the classes of $\{w \mapsto V^*_w(s,t)\}$ ▷ worlds grouped by their value at (s,t)
    else
      $\mathcal{C} \leftarrow \{\Theta\}$
    end if
    $\mathcal{P} \leftarrow \emptyset$ ▷ candidate sets
    for each action $a \in A$ **do**
      $G \leftarrow \{w \in \Theta : a \in A^*_w(s,t)\}$
      for each $C \in \mathcal{C}$ and each $N \in \mathcal{F}(T(s,a), t+1)$ **do**
        $P \leftarrow G \cap C \cap N$
        if $P \neq \emptyset$ **then**
          add P to $\mathcal{P}$ with record (a, N)
        end if
      end for
    end for
    $\mathcal{F}(s,t) \leftarrow \mathrm{maximal}(\mathcal{P})$ ▷ drop any candidate contained in another
  end for
end for
readout at the root: choose a minimum cover $\{M_1, \ldots, M_k\}$ of Θ by members of $\mathcal{F}(s_0, 0)$
assign each $w \in \Theta$ to one $M_i \ni w$; $\Sigma \leftarrow \{\sigma_1, \ldots, \sigma_k\}$
$\Lambda(\sigma_i) \leftarrow$ the action sequence read from the records $(a_0, N_1), (a_1, N_2), \ldots$ starting at M_i
return $\mathcal{F}$, $\langle \Sigma, \Lambda \rangle$

A.2 Proof of maximal SES at every node

For the following proof, we denote the SEB algorithm as $\mathcal{F}(s,t)$.

Proposition 2 (The backup finds the maximal Specification Equivalence Set at every node)**.** $\mathcal{F}(s,t) = \widehat{\mathcal{S}}(s,t)$ *at every reachable* (s,t).

Proof. By backward induction on t. W is a Specification Equivalence Set at (s,t) iff its members share an optimal first action a and W is also a Specification Equivalence Setat the next state $s_{t+1} = T(s,a)$. The first action of an optimal continuation attains the Bellman maximum in every $w \in W$, and conversely prepending an action optimal in every $w \in W$ to a common optimal continuation from s_{t+1} attains $V^*_w(s,t)$ in every w (the tail is one sequence because T is deterministic). The first condition is equivalent to $W \subseteq \{w : a \in BR_w(s,t)\}$, where $BR_w(s,t)$ is the set of optimal actions at (s,t) in world w. The second condition, because subsets of Specification Equivalence Set are Specification Equivalence Set (since they share a continuation), reads as $W \subseteq N$ for some $N \in \widehat{\mathcal{S}}(s_{t+1}, t+1)$ by the induction hypothesis. Therefore, the Specification Equivalence Set at (s,t) are exactly the subsets of the sets $\{w : a \in BR_w(s,t)\} \cap N$, meaning the maximal Specification Equivalence Set $\widehat{\mathcal{S}}(s,t)$ will be the maximal ones among those resulting in the algorithm detailed above. At $t = H-1$ there is no second condition and the Specification Equivalence Set are the subsets of the sets $\{w : a \in BR_w(s, H-1)\}$, which is the base case. □

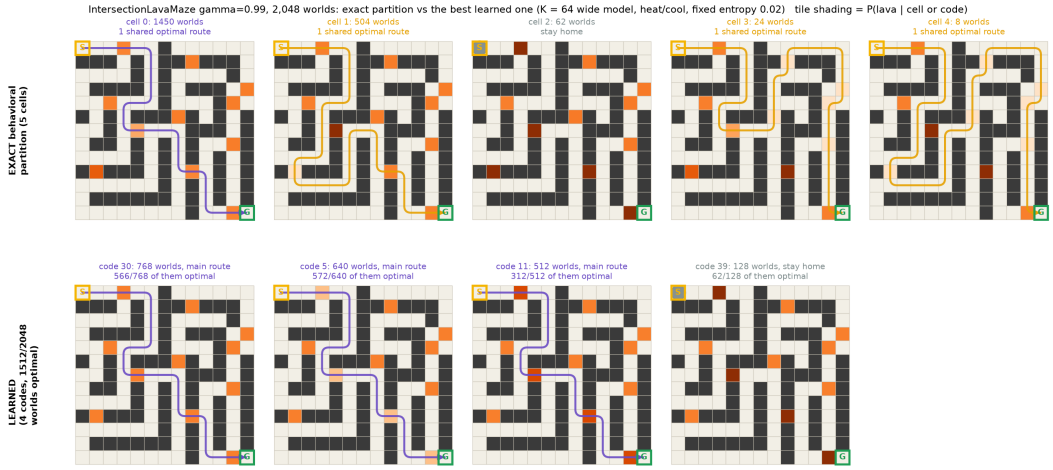

Figure 7: The top row shows the exact partitions for the 2048 possible parameterization UMDP, in which the color shows the frequency of lava being present in that set of levels. For example, cell 2 of the top row always has lava in the dark red spots. Along with this the optimal policy is also shown. The bottom row shows the best run for the ALSES version of the algorithm and its learned policy for its partitions of levels. Each maze is also labeled with the amount of optimal worlds within each set.

A.3 Valued SEB

The SEB algorithm outputs the minimal cover by maximal Specification Equivalence Sets in which every specification admits at least one optimal policy for all parameterizations in the group. But consider for instance, the case in which two specifications only differ by their reward function, one in which reaching the goal returns a large positive value, and in the other, that same goal instead gives that same value plus one. These two specifications will be in the same set even though, from the designer's perspective the value could be important to specify and thus would require two different sets. In order to do this we would modify the merging to also require the value to be equal between two nodes, ensuring that not only is the optimal policy the same, but the value of each node is the same.

While removing the value does give a coarser Minimal Complete Specification Language, the value-based version does solve a real use case, one in which the value of the problem has some external use or reward. An example of this might be in the case where two worlds have the same optimal policy but in one, along the way the agent must incur some large negative reward, in which case, the designer could want to actually differentiate between those cases and bail out beforehand. This is demonstrated in Section 7.3.

A.4 Experiments: Lava Maze

We run ALSES on the Lava Maze example seen in Figure 7 with the number of codes set as $k = 64$ and with a codebook usage penalty going from negative penalty to a positive one, encouraging the use of codes up until timestep 1500.

Even with a code usage bonus initial merges based on embedding are common. Similar worlds will embed near each other and partition together. Even in the case where the true MCSL is not found, the partition of the model recovers important policy aspects. For example, the model finds a go/don't go code describing worlds where its not worth going through lava and the agent learns to stay at home. Additionally, as seen in Figure 7 the partitions do learn interesting behavior about what the policy will encounter.

The model learns to partition into groups where lava is rarely encountered on the main path, while also finding groups where lava is found almost always, with the extreme being the stay-at-home par-

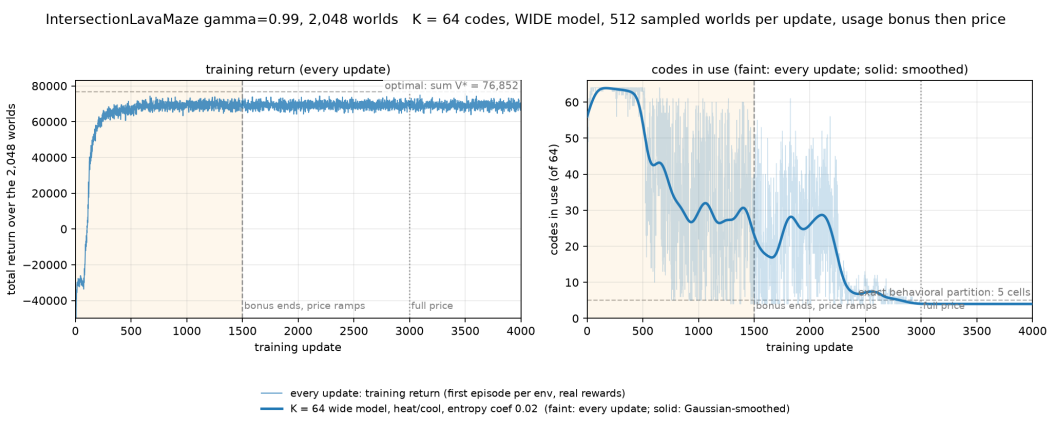

Figure 8: Between timestep 0 and timestep 1500 we slowly transition from a bonus term $\lambda = -0.01$ through 0 at timestep 1500 and 0.01 at timestep 3000. The dashed line on the left graph shows the optimal sum return in all the worlds. The right graph shows a Gaussian-smoothed code usage per timestep, with the full data faint behind it.

tition. We attribute this failure to exploration and code merging inhibiting the model from updating enough in the direction of a new found path. However, even when the path is non-optimal important specifications are still recovered about this unoptimal path. The training reward and the number of codes used over time can be seen in 8.

A.5 ALSES ON A NOISY UMDP FIGURES

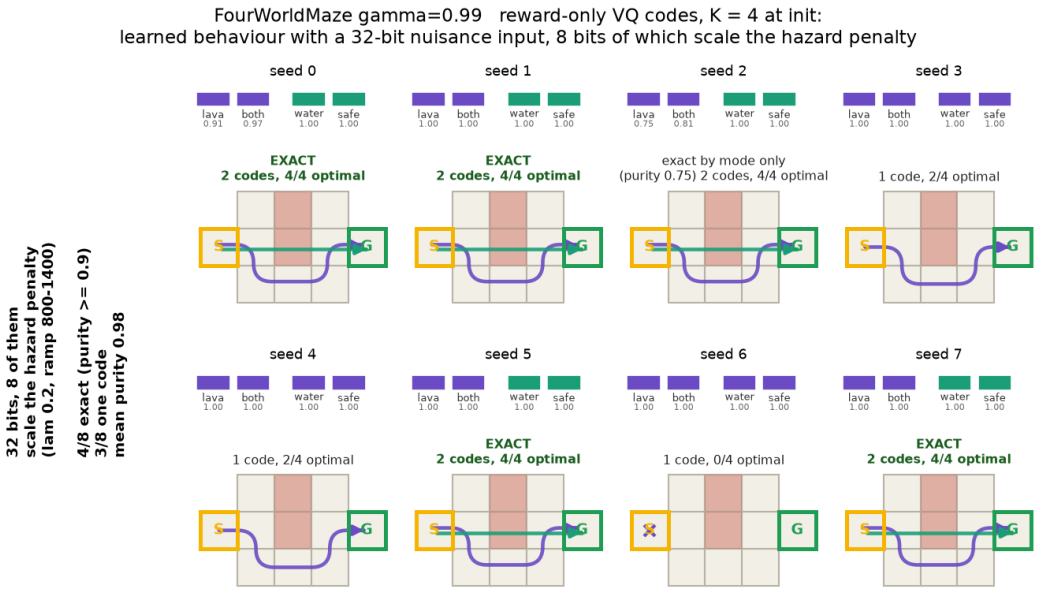

Figure 9: In this example the four mazes have an additional parameter: a 32bit integer which modifies the penalty of the lava and water as detailed in subsection 7.2. SEB struggles at these scales, but the ALSES method does recover the right partitioning of worlds in 4 of the 8 seeds.

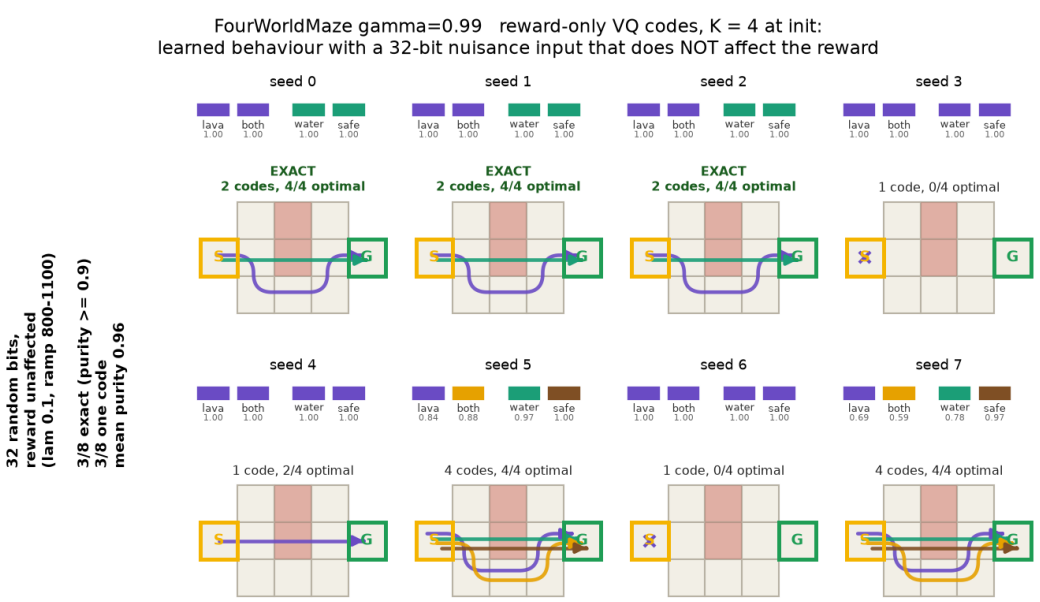

Figure 10: This is the same as Figure 9 but this time the 32bit integer supplied to the ALSES algorithm, does not affect anything within the world and is purely a label. Similarly to the other experiment, the ALSES method recovers the right partitions in 3/8 seeds, and finds optimal policies in 2 seeds, but uses 4 codes for it.

A.6 DAG OF FIGURE 2

Returning to Figure 2, and show what the output DAG would look like, after running SEB on it, and removing all nodes unnecessary for the final minimal cover Specification Equivalence Set.

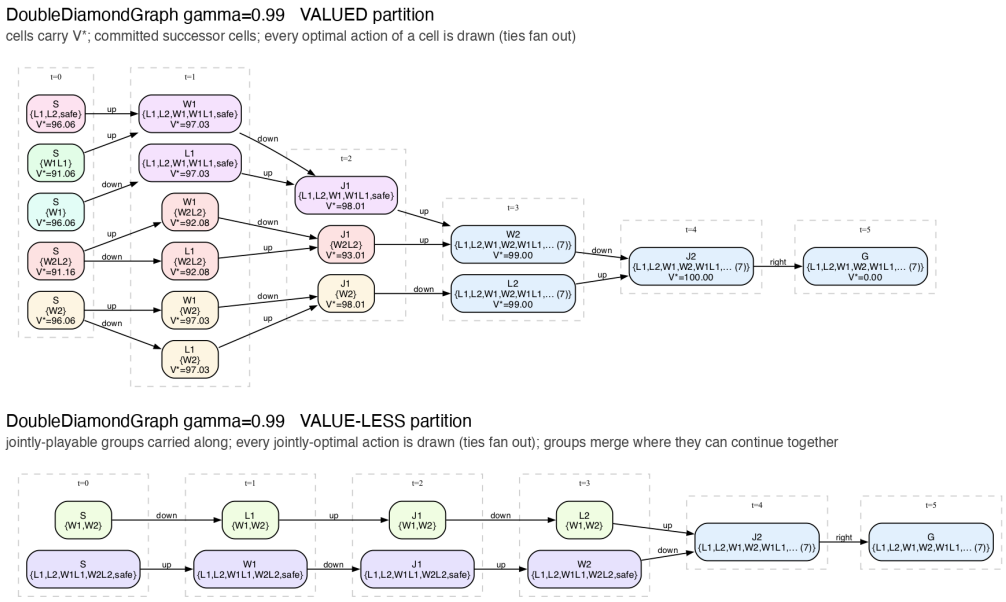

Figure 11: This figure demonstrates the resulting DAG from running the SEB method on the diamond maze from figure 2 but with all unecessary nodes removed for the MCSL in order to clear up the DAG. Each node is labeled with the state, within a timestep, and holds the Specification Equivalence Set for each level in each node. To find the optimal policy for a specification, we can follow the specific parameterization down the graph.