Answer-Set-Programming-based Abstractions for Reinforcement Learning

Rafael Bankosegger[1][0009-0003-3975-1012], Thomas Eiter[1][0000-0001-6003-6345], and Johannes Oetsch[2][0000-0002-9902-7662]

[1] Institute of Logic and Computation, TU Wien, Austria
[2] Department of Computing, Jönköping University, Sweden
rafael@bankosegger.at, thomas.eiter@tuwien.ac.at, johannes.oetsch@ju.se

Abstract. Reinforcement Learning (RL) enables autonomous agents to learn policies from experience, but realistic problems often involve enormous state spaces, making learning and generalisation challenging. Abstraction and approximation are therefore essential. Relational Reinforcement Learning (RRL) offers a way to reason about objects and their relations, and the CARCASS framework by Martijn van Otterlo demonstrates how logical representations can model Markov Decision Processes (MDPs) in first-order domains. Originally implemented in Prolog, CARCASS leverages domain knowledge to create powerful abstractions. As an alternative, we explore Answer-Set Programming (ASP) as an expressive and fully declarative modelling language for CARCASS. We evaluate our ASP-based implementation in case studies of two domains, viz. Blocks World and Minigrid. Our results indicate that CARCASS with ASP provides a promising approach to constructing abstractions for RL, especially when domain knowledge is available.[1]

Keywords: Relational Reinforcement Learning · Answer-Set Programming · State-Action-Space Abstractions.

1 Introduction

Reinforcement Learning (RL) [35] has emerged as a central paradigm in AI, providing a principled framework for agents to learn behaviour through interaction with an environment. By formalising sequential decision-making as a Markov Decision Process (MDP) [25], RL enables agents to improve their behaviour by trial and error, guided by the maximisation of cumulative reward [35]. Although this paradigm has produced impressive results, its practical application remains challenging: realistic problem settings often involve enormous state and action spaces, which render naive learning approaches intractable. Effective techniques for coping with this curse of dimensionality, such as abstraction and approximation, are therefore indispensable.

Relational Reinforcement Learning (RRL) builds on the use of first-order logic to introduce powerful abstractions for solving and reasoning about complex problem domains. The central idea is to extend the representations to a first-order setting, allowing for a more natural description of environments in terms of objects and their relations [38]. A major advantage of this approach is that it supports generalisation across similar states and even the transfer of learned knowledge to related tasks.

[1] Our implementation is available at https://github.com/rbankosegger/RLASP-core.

One particular approach to RRL, rooted in logic programming, is the CARCASS framework (short for *Computational Abstractions in Reinforcement Learning by using Conjunctive Abstractions of State Spaces*) introduced by Martijn van Otterlo [37]. CARCASS aims at abstracting the state-action space by representing abstract states as conjunctions of first-order literals, possibly enriched with background knowledge, and associates them with sets of admissible actions. An abstraction is thus an ordered list of rules of the form $\mathit{State} \rightarrow \{\text{Action}_1, \ldots, \text{Action}_n\}$, each capturing a situation described by state and the corresponding available actions. For example, the rule

$$\mathbf{cl}(X),\ \mathbf{cl}(Y),\ \mathbf{on}(Y,0) \quad \rightarrow \quad \{\mathbf{move}(X,Y),\ \mathbf{move}(Y,X)\}$$

represents an abstract blocks world state in which two blocks X and Y are clear and can be stacked on one another. A *policy* (what action to choose when in a given state) can then be learned via, e.g., Q-learning [40] over the abstract state-action space. One of CARCASS's key advantages is its ability to leverage domain knowledge in the abstraction rules, thereby reducing the effective size of the state–action space and supporting generalisation across related situations. The framework was originally realised in Prolog, making use of SLDNF-resolution to handle first-order reasoning over states and actions [37,39].

Building on this, we explore *Answer-Set Programming* (ASP) [2,11,20] as an alternative approach to revisiting CARCASS. ASP is a knowledge representation and reasoning framework with roots in logic programming, featuring a concise yet highly expressive modelling language which, unlike Prolog, is fully declarative. It supports nonmonotonic inference, reasoning under incomplete knowledge, expressing preferences and optimisation, as well as easy modelling of actions and change. These characteristics, combined with the ability to declaratively encode domain knowledge, make ASP particularly attractive for describing state abstractions in RRL. To the best of our knowledge, only two works so far use ASP in this context: a hierarchical framework in which ASP models high-level policies [21], and a robot architecture where ASP is applied to identify and eliminate irrelevant atoms in state descriptions, a form of state abstraction [34]. However, no ASP-based representation of a full state–action pair abstraction has yet been proposed.

In this work, we introduce a general method for encoding abstractions in ASP, which we evaluate using two case studies. The first is Blocks World, a classical planning problem that involves stacking blocks to reach a desired configuration; the second is MiniGrid, a suite of grid world navigation tasks, involving different subtasks such as key collection and door opening to reach a goal. Both domains have state spaces that are infeasibly large without abstraction.

Our main contribution can thus be summarised as follows:

(1) We introduce a general method for encoding CARCASS abstractions in ASP, demonstrating how a fully declarative and expressive modelling language can realise relational state–action abstractions;
(2) We show how to use ASP-based CARCASS abstractions for online learning; and
(3) We evaluate our ASP-based implementation in two case studies: Blocks World and MiniGrid.

Our results indicate that CARCASS with ASP provides a promising approach to constructing abstractions for RL, particularly when domain knowledge is available. The experiments show that, using Q-learning on the abstract representations, high-quality

policies can be learned consistently. These policies can also be obtained with significantly fewer samples than necessary without the abstraction.

The remainder of this paper is organised as follows. We start with preliminaries in Section 2. Section 3 presents our ASP-based approach for encoding CARCASS abstractions. Sections 4 and 5 describe the two case studies, Blocks World and MiniGrid, including an empirical evaluation with setup, results, and discussion in Section 6. Section 7 reviews related work, and Section 8 concludes with a discussion of implications and directions for future work.

For space reasons, some details are moved to the Appendix. Further material and discussion is available in [1].

2 Preliminaries

We next review relevant concepts from logic programming (SLDNF-resolution and ASP), reinforcement learning (MDP and Q-learning), and of the CARCASS framework.

2.1 Logic Programming

We assume a first-order language with constants c, functional terms $f(\bar{t})$, and predicate atoms $\mathbf{p}(\bar{t})$ over a PL1-signature $\Sigma = (\mathit{Func}, \mathit{Pred})$ and a set $\mathit{Var} = \{V_1, \ldots, V_m\}$ of variables, where $\bar{t} = t_1, \ldots, t_n$ is a list of terms of matching the arity of f resp. p.

Normal Logic Programs, SLDNF-Resolution. A *naf-literal* with negation as failure is a formula *not* $\mathbf{p}(\bar{t})$, where $\mathbf{p}(\bar{t})$ is an atom. A *normal rule* is of the form $h \leftarrow b_1, \ldots, b_n.$, where h is an atom and $b_1, \ldots, b_n$ are naf-literals. A *normal program* $P = \{r_1, \ldots, r_n\}$ is a set of normal rules. The application of a *substitution* $\theta : \mathit{Var} \to \mathit{Term}$ from a set of variables to a set of terms is defined as usual. Syntactic objects without variables are called *ground*. Given a program P and a *normal goal* $\leftarrow b_1, \ldots, b_n.$, an *SLDNF-refutation* of $P \cup \{\leftarrow b_1, \ldots, b_n.\}$ is denoted by $P \vdash_{\text{SLDNF}} b_1, \ldots, b_n$ [24, Chapter 8].

Herbrand Interpretations. Based on a signature $\Sigma = (\mathit{Func}, \mathit{Pred})$ we denote by $\mathcal{U}(\Sigma)$ the Herbrand universe over *Func*, by $\mathcal{B}(\Sigma)$ the Herbrand base over $\mathcal{U}(\Sigma)$ and *Pred*, and by $\mathcal{I}(\Sigma) = 2^{\mathcal{B}(\Sigma)}$ the set of Herbrand interpretations. An interpretation $I \in \mathcal{I}(\Sigma)$ is to be read such that an atom $\mathbf{p}(\bar{t})$ is *true* if $\mathbf{p}(\bar{t}) \in I$ and *false* if $\mathbf{p}(\bar{t}) \notin I$.

Answer-Set Programming (ASP). We consider ASP programs as defined in the *ASP-Core-2 input language format* [3], with the usual features such as strong negation, disjunctive rule heads, and optimisation. In particular, our encodings make use of choice rules, aggregates, and weak constraints. The set of all answer sets of a program P is denoted by $\mathcal{AS}(P) = \{I, J, \ldots\}$.

2.2 Reinforcement Learning

Reinforcement Learning [35] is a discrete-time, stochastic control process governed by the dynamics of a *task environment* and by the actions of a learning *agent*. At every time point t, the agent perceives the current *state* S_t of the environment and performs an

action A_t. The environment state is changed to a new state S_{t+1} based on the effects of the action and a *reward* R_{t+1} is obtained, acting as a performance measure. The result is a history of interactions $\mathsf{H} = (\mathsf{S}_0, \mathsf{A}_0, \mathsf{R}_1, \mathsf{S}_1, \mathsf{A}_1, \mathsf{R}_2, \mathsf{S}_2, \ldots)$.

Definition 1 (Markov Decision Process (MDP) [35,26]). *The task environment is formalised as (*finite*)* Markov decision process $M = (S, A, R, \Psi, p, \gamma)$*, where S is a finite set of states; A is a finite set of actions; $R \subset \mathbb{R}$ is finite; $\Psi \subseteq S \times A$ is the set of admissible state-action pairs, with $A_s = \{a \in A \mid (s, a) \in \Psi\}$ denoting the set of actions that are* admissible *in state s; $p : S \times R \times \Psi \to [0, 1]$, denoted as $p(s', r \mid s, a)$, defines the* dynamics *of the environment, i.e., the probability of transitioning to state s' with a reward of r after performing action a in state s; and $\gamma \in [0, 1]$ is the discount rate.*

Definition 2 (Policy [35,26]). *A* policy $\pi : \Psi \to [0, 1]$*, denoted by $\pi(a \mid s)$, is a probability distribution that models the decision making of the agent, i.e., the probability of choosing an admissible action $a \in A_s$ in state $s \in S$.*

Given an MDP $M = (S, A, R, \Psi, p, \gamma)$ and a policy π, the agent-environment interaction is defined by $\mathsf{A}_t \sim \pi(\,\cdot \mid \mathsf{S}_t)$ and $\mathsf{S}_{t+1}, \mathsf{R}_{t+1} \sim p(\,\cdot \mid \mathsf{S}_t, \mathsf{A}_t)$. The performance of the agent is measured in terms of the *γ-discounted return* $\mathsf{G}_t \doteq \sum_{k=0}^{\infty} \gamma^k \mathsf{R}_{t+k+1}$. The goal is to find an optimal policy $\pi_\star$ that maximises the expected return for all states, that is

$$v_\star(s) = \mathbb{E}_{p,\pi_\star}\left[\mathsf{G}_t \mid \mathsf{S}_t = s\right] \geq \mathbb{E}_{p,\pi}\left[\mathsf{G}_t \mid \mathsf{S}_t = s\right] = v_\pi(s) \quad \forall \pi, s, t.$$

Definition 3 (Q-Learning [40]). *A classic method to find $\pi_\star$ is the value-based Q-*learning *algorithm. Given a step-size parameter $\alpha \in (0, 1]$ and an action-value function $\mathsf{Q} : \Psi \to \mathbb{R}$, the version we use [35, Chapter 6.5] is governed by the update:*

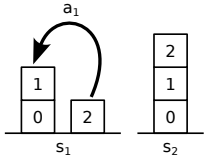
$$\mathsf{Q}(\mathsf{S}_t, \mathsf{A}_t) \leftarrow \mathsf{Q}(\mathsf{S}_t, \mathsf{A}_t) + \alpha \cdot \left(\mathsf{R}_{t+1} + \gamma \cdot \max_a \{\mathsf{Q}(\mathsf{S}_{t+1}, a)\} - \mathsf{Q}(\mathsf{S}_t, \mathsf{A}_t)\right)$$

It was shown that, under specific conditions such as continued exploration of all state-action pairs and proper management of α, $\mathsf{Q}(s, a)$ converges with probability 1 to $q_\star(s, a) = \mathbb{E}_{p,\pi_\star}\left[\mathsf{G}_t \mid \mathsf{S}_t = s \wedge \mathsf{A}_t = a\right]$ from arbitrary initial values for Q and independent of the *behaviour policy* used to select the actions [40,36,15]. The learned *target policy* $\pi_\star$ can be approximated by selecting actions based on $\arg\max\{\mathsf{Q}(s, a) \mid a \in A_s\}$.

2.3 Relational MDP and CARCASS Abstractions

We now introduce the CARCASS framework of Otterlo [39, Chapter 5]. We start with refining MDPs to relational MDPs. Then we define abstract states and actions, followed by the CARCASS syntax and semantics, and finally an abstract version of Q-learning.

Definition 4 (Relational MDP [39, p. 168]). *A* relational Markov decision process (RMDP) *is a sixtuple $M' = (\Sigma, S, A, R, \Psi, p, \gamma)$ where $\Sigma = (Func, Pred_S \cup Pred_A)$ is a PL1-signature, $S \subseteq \mathcal{I}((Func, Pred_S))$ and $A \subseteq \mathcal{B}((Func, Pred_A))$ are finite state and action sets, respectively, and $M = (S, A, R, \Psi, p, \gamma)$ is an MDP. States are Herbrand interpretations. We define the logic programs (i.e. sets of facts) $P_s \doteq \{\, \boldsymbol{p}(\bar{t}). \mid \boldsymbol{p}(\bar{t}) \in s \,\}$ and $P_{A_s} \doteq \{\, \boldsymbol{p}(\bar{t}). \mid \boldsymbol{p}(\bar{t}) \in A_s \,\}$.*



$$s_1 = \left\{ \begin{array}{l} \mathbf{on}(0, table), \mathbf{on}(1, 0), \mathbf{on}(2, table), \\ \mathbf{goal}(0, table), \mathbf{goal}(1, 0), \mathbf{goal}(2, 1) \end{array} \right\}$$

$$a_1 = \mathbf{move}(2, 1)$$

$$s_2 = \left\{ \begin{array}{l} \mathbf{on}(0, table), \mathbf{on}(1, 0), \mathbf{on}(2, 1), \\ \mathbf{goal}(0, table), \mathbf{goal}(1, 0), \mathbf{goal}(2, 1) \end{array} \right\}$$

Fig. 1. Example states from a blocks world environment.

Example 1 (Blocks World RMDP). Consider a blocks world stacking task [33] with three blocks, as illustrated in Fig. 1. The state space S is the set of all 13 valid stacking configurations as denoted by **on**/2. The goal description, denoted by **goal**/2, stays constant throughout the state space. The action space A contains all the possible moves, denoted by **move**/2. The admissibility of moves is restricted to those that make sense for any given configuration. For example, $A_{s_1} = \{\mathbf{move}(2, 1), \mathbf{move}(1, 2), \mathbf{move}(1, table)\}$. The transitions are deterministic, defined to be consistent with the usual effects of moves. The reward is 99 if a goal state is reached, -1 otherwise, and $\gamma = 1$. For example, performing the action a_1 in state s_1 yields a new state s_2 and a reward of 99, since s_2 matches the goal description.

Definition 5 (Abstract States and Actions [39, p. 253]). *Let Σ be a PL1-Signature and Var a set of variables. An* abstract state *$\hat{s}$ is of the form $\hat{s} \doteq l_1, \ldots, l_k$, such that $k \geq 1$ and every l_i is a naf-literal over Σ and Var.*[2] *An* abstract action *$\hat{a}$ is a single predicate atom $\hat{a} \doteq \boldsymbol{p}(\bar{t})$ over Σ and Var.*

Definition 6 (CARCASS Syntax and Semantics [39, p. 253]). *Let Σ be a PL1-signature and Var a set of variables. A* CARCASS *$\hat{C} = \langle(\hat{s}_1, \hat{A}_{\hat{s}_1}), \ldots, (\hat{s}_n, \hat{A}_{\hat{s}_n})\rangle$ is a finite ordered list of $n \geq 1$ elements, such that $\hat{s}_1, \ldots, \hat{s}_n$ are abstract states and $\hat{A}_{\hat{s}_1}, \ldots, \hat{A}_{\hat{s}_n}$ are sets of abstract actions over Σ and Var. Every $A_{\hat{s}_i} = \{\hat{a}_{i,1}, \ldots, \hat{a}_{i,m_i}\}$ denotes the set of abstract actions that are* admissible *in the abstract state $\hat{s}_i$. For every $1 \leq i \leq n$ and $1 \leq j \leq m_i$, the variables occurring in $\hat{a}_{i,j}$ must also occur in $\hat{s}_i$. Furthermore, let $\hat{S} = \{\hat{s}_i \mid 1 \leq i \leq n\}$ and $\hat{\Psi} = \{(\hat{s}_i, \hat{a}_{i,j}) \mid 1 \leq i \leq n, 1 \leq j \leq m_i\}$. Now let $M = (\Sigma, S, A, R, \Psi, p, \gamma)$ be an RMDP of matching signature. For every $\hat{s}_i \in \hat{S}$ and $(\hat{s}_i, \hat{a}_{i,j}) \in \hat{\Psi}$ we define:*

$$\begin{aligned} Cov(\hat{s}_i) &\doteq \{\ s \in S \mid P_s \vdash_{SLDNF} \hat{s}_i\ \} \\ Cov(\hat{s}_i, \hat{a}_{i,j}) &\doteq \{\ (s, a) \in \Psi \mid P_s \vdash_{SLDNF} \hat{s}_i\theta \text{ and } a = \hat{a}_{i,j}\theta\ \} \\ [\![\hat{s}_i]\!]_{\hat{C}} &\doteq \{\ s \in Cov(\hat{s}_i) \mid s \notin Cov(\hat{s}_k) \text{ for all } 1 \leq k < i\ \} \\ [\![\hat{s}_i, \hat{a}_{i,j}]\!]_{\hat{C}} &\doteq \{\ (s, a) \in Cov(\hat{s}_i, \hat{a}_{i,j}) \mid s \notin Cov(\hat{s}_k) \text{ for all } 1 \leq k < i\ \} \\ [\![\hat{s}_i, \hat{a}_{i,j}]\!]^s_{\hat{C}} &\doteq \{\ a \mid (s, a) \in [\![\hat{s}_i, \hat{a}_{i,j}]\!]_{\hat{C}}\ \} \end{aligned}$$

Example 2. An example CARCASS for a 3-blocks world is given in Table 1. Intuitively, $\hat{s}_1$ captures all states with a tower of two blocks, $\hat{s}_2$ captures all states with all blocks on the table, and $\hat{s}_3$ captures all states with a tower of three blocks.

[2] This includes built-in atoms. Moving away from Prolog, strong negation and aggregates may also be permitted.

Table 1. Example CARCASS [39, p. 253]

$\hat{s}_1$:	$\{\mathbf{on}(A,B), \mathbf{on}(B,\mathit{table}), \mathbf{on}(C,\mathit{table}), A \neq B, B \neq C\}$
$\hat{a}_{1,1}, \hat{a}_{1,2}, \hat{a}_{1,3}$:	$\mathbf{move}(A,C), \mathbf{move}(C,A), \mathbf{move}(A,\mathit{table})$
$\hat{s}_2$:	$\{\mathbf{on}(A,\mathit{table}), \mathbf{on}(B,\mathit{table}), \mathbf{on}(C,\mathit{table}), A \neq B, B \neq C, A \neq C\}$
$\hat{a}_{2,1}, \ldots, \hat{a}_{2,6}$:	$\mathbf{move}(A,B), \mathbf{move}(B,A), \mathbf{move}(A,C),$ $\mathbf{move}(C,A), \mathbf{move}(B,C), \mathbf{move}(C,B)$
$\hat{s}_3$:	$\{\mathbf{on}(A,B), \mathbf{on}(B,C), \mathbf{on}(C,\mathit{table}), A \neq B, B \neq C, C \neq \mathit{table}\}$
$\hat{a}_{3,1}$:	$\mathbf{move}(A,\mathit{table})$

Algorithm 1.1: Q-learning for CARCASSs [39, p. 258]

Input: RMDP $M = (\Sigma, S, A, R, \Psi, p, \gamma)$, CARCASS $\hat{C} = \langle(\hat{s}_1, \hat{A}_{\hat{s}_1}), \ldots, (\hat{s}_n, \hat{A}_{\hat{s}_n})\rangle$, initial abstract action-value function $\hat{\mathsf{Q}} : \hat{\Psi} \to \mathbb{R}$

Output: Learned abstract action-value function $\hat{\mathsf{Q}}$ (ideally $\hat{\mathsf{Q}} \approx q_\star$)

foreach episode **do**
  Initialize the starting state S_0
  Find $\hat{\mathsf{S}}_0 \in \hat{S}$ for which $\mathsf{S}_0 \in [\![\hat{\mathsf{S}}_0]\!]_{\hat{C}}$
  repeat for every time point $t = 0, 1, 2, \ldots$
    Choose $\hat{\mathsf{A}}_t \in \hat{A}_{\hat{\mathsf{S}}_t}$ using a behaviour policy derived from $\hat{\mathsf{Q}}$
    Choose a random action $\mathsf{A}_t \in [\![\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t]\!]^{\mathsf{S}_t}_{\hat{C}}$
    Take action A_t, observe $\mathsf{S}_{t+1}, \mathsf{R}_{t+1}$
    Find $\hat{\mathsf{S}}_{t+1} \in \hat{S}$ for which $\mathsf{S}_{t+1} \in [\![\hat{\mathsf{S}}_{t+1}]\!]_{\hat{C}}$
    $\hat{\mathsf{Q}}\left(\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t\right) \leftarrow \hat{\mathsf{Q}}\left(\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t\right) + \alpha \cdot \left(\mathsf{R}_{t+1} + \gamma \cdot \max_{\hat{a}} \left\{\hat{\mathsf{Q}}\left(\hat{\mathsf{S}}_{t+1}, \hat{a}\right)\right\} - \hat{\mathsf{Q}}\left(\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t\right)\right)$
  until *end of episode* **or** *time limit reached*
end

A CARCASS has two aspects: a covering relation *Cov*, defined via SLDNF-resolution, and a decision list. Intuitively, to choose an abstract state-action pair for some $(s, a) \in \Psi$, this list can be traversed until the first $\hat{s}_i$ is found which covers s. Then, $\hat{A}_{\hat{s}_i}$ can be searched for some $\hat{a}_{i,j}$ such that that $(\hat{s}_i, \hat{a}_{i,j})$ covers (s, a).

Abstract Q-Learning. Algorithm 1.1 illustrates how Q-learning can operate on the abstract representations of a CARCASS. The algorithm assumes that the task environment is explored in a series of episodes, each a history of interactions $\mathsf{H} = (\mathsf{S}_0, \mathsf{A}_0, \mathsf{R}_1, \mathsf{S}_1, \ldots)$ on the concrete level. In addition, the CARCASS semantics is used to derive an abstract series of interactions $\hat{\mathsf{H}} = (\hat{\mathsf{S}}_0, \hat{\mathsf{A}}_0, \mathsf{R}_1, \hat{\mathsf{S}}_1, \ldots)$, to which Q-learning is applied, working with an abstract action-value function $\hat{\mathsf{Q}} : \hat{\Psi} \mapsto \mathbb{R}$. Abstract actions are chosen based on an abstract policy $\hat{\pi} : \hat{\Psi} \to [0, 1]$.

A number of results establish the *convergence* of abstract Q-learning under certain conditions, such as when the behaviour policy is stationary [32] or if the abstraction maintains certain properties of the concrete MDP, like $q_\star$*-irrelevance* for state abstractions [19] or *MDP homomorphisms* [26,27] for state-action pair abstractions. Although there are no convergence guarantees in general (and even examples of failure [19]), Sutton

and Barto [35, p. 263] conjectured that such guarantees exist if the behaviour policy is sufficiently close to the target policy, pointing out that they are not aware of any observed cases of divergence for this case.

3 ASP-Based Abstractions

We provide a method to encode CARCASS abstractions in ASP and show how the resulting ASP encoding can be used in an online learning setting. We also extend the method to allow for additional background knowledge.

3.1 ASP Encoding

Let $\hat{C} = \langle(\hat{s}_1, \hat{A}_{\hat{s}_1}), \ldots, (\hat{s}_n, \hat{A}_{\hat{s}_n})\rangle$ be a CARCASS and $M = (\Sigma, S, A, R, \Psi, p, \gamma)$ an RMDP with a shared signature $\Sigma = (Func, Pred_S \cup Pred_A)$. We assume that none of the function- and predicate names introduced below occur in Σ.

Definition 7 (Labelling Function). *To reason about abstract states and state-action pairs as objects, we define a* labelling function *as a one-to-one mapping $\lambda : \hat{S} \cup \hat{\Psi} \mapsto Term$ from abstract states and state-action pairs to a set of ground terms.*

Definition 8 (CARCASS ASP Encoding). *Let $\hat{s}_i \in \hat{S}$ be an abstract state and let $\bar{V} = V_1, \ldots, V_m$ denote all the variables occurring in $\hat{s}_i$. Then, $\hat{s}_i$ is encoded as $P_{\hat{s}_i}$. An abstract pair $(\hat{s}_i, \hat{a}_{i,j}) \in \hat{\Psi}$ with $\hat{a}_{i,j} = \boldsymbol{p}(\bar{t})$ is encoded as $P_{\hat{s}_i,\hat{a}_{i,j}}$, containing a newly introduced function name p/n with the same arity as the predicate name $\boldsymbol{p}/n$. The covering relation is encoded by P_{Cov} and the full CARCASS semantics by $P_{\hat{C}}$.*

$$P_{\hat{s}_i} \doteq \left\{ \boldsymbol{sIdx}(\lambda(\hat{s}_i), i). \qquad \boldsymbol{sCov}(\lambda(\hat{s}_i), (\bar{V})) \leftarrow \hat{s}_i. \right\}$$

$$P_{\hat{s}_i,\hat{a}_{i,j}} \doteq \left\{ \boldsymbol{aCov}(\lambda(\hat{s}_i, \hat{a}_{i,j}), p(\bar{t})) \leftarrow \boldsymbol{sChoice}(\lambda(\hat{s}_i)), \boldsymbol{sCov}(\lambda(\hat{s}_i), (\bar{V})), \boldsymbol{p}(\bar{t}). \right\}$$

$$P_{Cov} \doteq \bigcup_{1 \le i \le n} \left(P_{\hat{s}_i} \cup \bigcup_{1 \le j \le m_i} P_{\hat{s}_i,\hat{a}_{i,j}} \right) \cup \left\{ 1 = \{ \boldsymbol{sChoice}(R) : \boldsymbol{sCov}(R, \_) \}. \right\}$$

$$P_{\hat{C}} \doteq P_{Cov} \cup \left\{ \begin{array}{l} \leftarrow \boldsymbol{sChoice}(R1), \boldsymbol{sCov}(R2, \_), \\ \quad \boldsymbol{sIdx}(R1, I1), \boldsymbol{sIdx}(R2, I2), I2 < I1. \end{array} \right\}$$

Example 3. Consider the abstract pair $(\hat{s}_2, \hat{a}_{2,1})$ from Example 2. We choose descriptive names as labels, e.g. $\lambda(\hat{s}_2) =$ `"alltable"` and $\lambda((\hat{s}_2, \hat{a}_{2,1})) =$ `"mAB"`. Then:

$$P_{\hat{s}_2} = \left\{ \begin{array}{l} \mathbf{sIdx}(\texttt{"alltable"}, 2). \\ \mathbf{sCov}(\texttt{"alltable"}, (A, B, C)) \leftarrow \mathbf{on}(A, \mathit{table}), \mathbf{on}(B, \mathit{table}), \mathbf{on}(C, \mathit{table}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad A \neq B, B \neq C, A \neq C. \end{array} \right\}$$

$$P_{\hat{s}_2,\hat{a}_{2,1}} = \left\{ \begin{array}{l} \mathbf{aCov}(\texttt{"mAB"}, \mathit{move}(A, B)) \leftarrow \mathbf{sChoice}(\texttt{"alltable"}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{sCov}(\texttt{"alltable"}, (A, B, C)), \mathbf{move}(A, B). \end{array} \right\}$$

The encoding introduces several new atoms.

- **sIdx**$(\lambda(\hat{s}_i), i)$ states the existence of an abstract state $\hat{s}_i$ (identified by label) and its index i in the decision list.
- **sCov**$(\lambda(\hat{s}_i), (\bar{T}))$ states the existence of a covering relation between $\hat{s}_i$ and some state s. The tuple $(\bar{T})$ represents a ground substitution $\bar{V}\theta = \bar{T}$, reminiscent of the original definition of *Cov*, where $P_s \vdash_{\text{SLDNF}} \hat{s}_i\theta$.
- **sChoice**$(\lambda(\hat{s}_i))$ states $\hat{s}_i$ as "chosen", such that all abstract states covering some concrete state can be enumerated as answer sets (by the choice rule in P_{Cov}).
- **aCov**$(\lambda(\hat{s}_i, \hat{a}_{i,j}), p(\bar{t}))$ states the existence of a covering relation between an abstract pair $(\hat{s}_i, \hat{a}_{i,j})$ and a concrete pair (s, a) in an answer set where $\hat{s}_i$ is "chosen".

Correctness. We can establish the following proposition (proof in the appendix):

Proposition 1. *With CARCASS* $\hat{C} = \langle(\hat{s}_1, \hat{A}_{\hat{s}_1}), \ldots, (\hat{s}_n, \hat{A}_{\hat{s}_n})\rangle$*, RMDP* $M = (\Sigma, S, A, R, \Psi, p, \gamma)$ *of matching signature, and labels* $\lambda : \hat{S} \cup \hat{\Psi} \mapsto$ *Term, let* $s \in S$*,* $P = P_s \cup P_{A_s} \cup P_{Cov}$*,* $I \in \mathcal{AS}(P)$*, and* $\hat{s}_i = l_1, \ldots, l_k \in \hat{S}$ *with variables* $V_1, \ldots, V_m$*. Further let* θ *be a substitution such that* $\hat{s}_i\theta$ *is a ground instance of* $\hat{s}_i$*. Then,* $P_s \vdash_{SLDNF} \hat{s}_i\theta$ *iff* ***sCov***$(\lambda(\hat{s}_i), (V_1\theta, \ldots, V_m\theta)) \in I$*.*

The following properties (P1)-(P3) of the translation are then established:

(P1) The answer sets $\mathcal{AS}(P_s \cup P_{A_s} \cup P_{Cov})$ correspond one-to-one with the abstract states that cover $s \in S$.

(P2) Each such answer set, corresponding with the chosen abstract state $\hat{s}_i$, contains a listing of the admissible abstract actions $\hat{a}_{i,j} \in \hat{A}_{\hat{s}_i}$ and their covered admissible concrete actions $\mathbf{p}(\bar{t}) = a \in A_s$, in the form of **aCov**$(\lambda(\hat{s}_i, \hat{a}_{i,j}), \mathrm{p}(\bar{t}))$ atoms.

(P3) $\mathcal{AS}(P_s \cup P_{A_s} \cup P_{\hat{C}})$ contains at most one such answer set, corresponding to the covering abstract state with the lowest index.

Property (P1) follows from the choice rule in P_{Cov}. To verify (P2), consider the rule

$$\mathbf{aCov}(\lambda(\hat{s}_i, \hat{a}_{i,j}), \mathrm{p}(\bar{t})) \leftarrow \mathbf{sChoice}(\lambda(\hat{s}_i)), \mathbf{sCov}(\lambda(\hat{s}_i), (\bar{V})), \mathbf{p}(\bar{t}).$$

in $P_{\hat{s}_i, \hat{a}_{i,j}}$, which has a ground instance for every occurrence of **sCov** in $I \in \mathcal{AS}(P_s \cup P_{A_s} \cup P_{Cov})$. The body of this ground instance will be true if

- $\hat{s}_i$ is "chosen" in I (by the choice rule in P_{Cov}),
- the substitution θ related with the ground instance is such that $P_s \vdash_{\text{SLDNF}} \hat{s}_i\theta$, and
- there exists an admissible concrete action $\mathbf{p}(\bar{t})\theta = a \in A_s$.

These criteria correspond with the ones from the definition of $Cov(\hat{s}_i, \hat{a}_{i,j})$ and the rule head rightly asserts that $(\hat{s}_i, \hat{a}_{i,j})$ covers (s, a). Generalising this argument, we can see how $P_{\hat{s}_i, \hat{a}_{i,j}}$ identifies all admissible actions for which the covering relation holds.

To verify (P3), observe that $P_{\hat{C}}$ is just P_{Cov} plus a constraint, eliminating all answer sets except the one where the index of the chosen covering abstract state is minimal.

3.2 Online Learning for ASP-Based CARCASS Abstractions

Algorithm 1.2 illustrates how the ASP-encoding of a CARCASS can be used in an online learning setting such as Algorithm 1.1. The ASP encoding used here includes the programs discussed so-far, in addition a program P_B with background knowledge, and the program

$$P_{\hat{s}_\infty} \doteq \left\{ \begin{array}{l} \mathbf{cState}(\texttt{"true"}, \#sup). \\ \mathbf{cStateCovers}(\texttt{"true"}, ()). \end{array} \right\}$$

Algorithm 1.2: Online interaction for ASP-encoded CARCASSs

Input: Current state s with admissible actions A_s, background knowledge P_B, and a CARCASS $\hat{C} = \langle(\hat{s}_1, \hat{A}_{\hat{s}_1}), \ldots, (\hat{s}_n, \hat{A}_{\hat{s}_n})\rangle$ with labels λ and encoding $P_{\hat{C}}$

Output: Chosen abstract state $\hat{s}$, actions $\hat{A}'_{\hat{s}} \subseteq \hat{A}_{\hat{s}}$, and $[\![\hat{s}, \hat{a}]\!]^s_{\hat{C}}$ for all $\hat{a} \in \hat{A}'_{\hat{s}}$.

1 $I \leftarrow$ An arbitrarily chosen optimal answer set $I \in \mathcal{AS}(P_{\hat{C}} \cup P_s \cup P_{A_s} \cup P_{\hat{s}_\infty} \cup P_B)$
2 $\hat{s} \leftarrow \lambda^{-1}(l)$ for the only $\mathbf{sChoice}(l) \in I$
3 **if** $\hat{s} = \hat{s}_\infty$ **then**
4 $\quad \hat{A}'_{\hat{s}} \leftarrow \{\hat{a}_{\infty,1}\}$
5 $\quad [\![\hat{s}, \hat{a}_{\infty,1}]\!]^s_{\hat{C}} \leftarrow A_s$
6 **else**
7 $\quad \hat{A}'_{\hat{s}} \leftarrow \{\hat{a} \mid \mathbf{aCov}(l, p(\bar{t})) \in I \wedge \lambda(\hat{s}, \hat{a}) = l\}$
8 $\quad$ **for** $\hat{a} \in \hat{A}'_{\hat{s}}$ **do**
9 $\quad\quad [\![\hat{s}, \hat{a}]\!]^s_{\hat{C}} \leftarrow \{\mathbf{p}(\bar{t}) \mid \mathbf{aCov}(l, p(\bar{t})) \in I \wedge \lambda(\hat{s}, \hat{a}) = l\}$
10 $\quad$ **end**
11 **end**

representing an empty abstract state $\hat{s}_\infty = \top$ that trivially covers every concrete state and always has the highest index. The admissible actions are defined as $\hat{A}_{\hat{s}_\infty} \doteq \{\hat{a}_{\infty,1}\}$, such that $[\![\hat{s}_\infty, \hat{a}_{\infty,1}]\!]^s_{\hat{C}} \doteq A_s$ covers all admissible concrete actions. In line 1 of the algorithm, the augmented ASP encoding is handed to an ASP solver, which is expected to return an optimal answer set. If $P_B = \emptyset$, there exists only a single answer set corresponding to the covering abstract state with the smallest index, as previously described. If there are multiple optimal answer sets, the choice of which to return is left to the solver. Lines 3–5 are then concerned with the special case in which $\hat{s}_\infty$ was chosen, and lines 7–10 deal with the extraction of information from the returned answer set in case some other $\hat{s} \in \hat{S}$ was chosen. The algorithm returns the chosen abstract state $\hat{s}$, a set of abstract actions $\hat{A}'_{\hat{s}} \subseteq \hat{A}_{\hat{s}}$ that cover at least one admissible concrete action, to be used instead of $\hat{A}_{\hat{s}}$ in the definition of abstract policies, and the sets $[\![\hat{s}, \hat{a}]\!]^s_{\hat{C}}$ for all $\hat{a} \in \hat{A}'_{\hat{s}}$.

Background Knowledge. The algorithm allows that the CARCASS encoding is combined with additional background knowledge in the form of an ASP program P_B, including the possibility to add weak constraints. However, if P_B causes the generation of multiple (optimal) answer sets, one needs to be careful that they agree on the chosen abstract state. A simple way to ensure this is to add the weak constraint $:\sim \mathbf{sChoice}(R), \mathbf{sIdx}(R, I).\ [I@1, R]$ to act as a tie breaker on the lowest priority level, provided that other weak constraints are defined only on priority levels 2 or higher.

4 Case Study: Blocks World

As a first application, we discuss n-blocks world stacking tasks [33] and present the encoding of a state-action pair abstraction, inspired by the algorithms GN1 and GN2 [33], in which the size of the abstract state-action space is constant in the number of blocks. As a trade-off, the abstraction does not preserve the optimal policy for the (NP-hard) blocks world planning problem. We only glance over some of the details here; the full CARCASS encoding is available in Appendix B.

Blocks World as RMDP. We assume a standard n-blocks world setting with a *table* and $n \geq 1$ blocks as objects. A state is described using the predicates $\mathbf{on}(B, L)$ to denote that block B is on top of location L; and $\mathbf{goal}(B, L)$ to denote that B must be on top of L in a goal state. All actions are based on $\mathbf{move}(B, L)$, denoting the move of block B to some new location L. The effects of moves are deterministic and according to the standard blocks world dynamics. The reward is 99 when the last action caused a transition to a goal state, i.e. $\mathbf{on}(B, L)$ holds true for each stated $\mathbf{goal}(B, L)$. Note that partial goal descriptions are allowed. Otherwise, the reward is -1. The discount rate is $\gamma = 1$.

Background Knowledge. Several predicates were defined to support the abstraction, including the following: $\mathbf{cl}(L)$ denotes that location L is clear, i.e., no blocks are located on top, or L is the table; $\mathbf{above}/2$ denotes the transitive closure of $\mathbf{on}/2$; and $\mathbf{incomplete}(L, B)$ denotes the existence of an *incomplete tower* in its final position, with location L, topmost block B, and other misplaced blocks allowed to occur on top of B; incompleteness means that some further block(s) must be added to reach a goal state.

Example 4 (Example 1 cont'd). We augment s_1 with the facts $\mathbf{cl}(table)$, $\mathbf{cl}(1)$, $\mathbf{cl}(2)$, $\mathbf{above}(0, table)$, $\mathbf{above}(1, 0)$, $\mathbf{above}(1, table)$, $\mathbf{above}(2, table)$, and $\mathbf{incomplete}(table, 1)$. To further illustrate incomplete towers, consider

$$s_3 = \left\{ \begin{array}{l} \mathbf{on}(4, table), \mathbf{on}(0, 4), \mathbf{on}(1, 0), \mathbf{on}(2, 1), \mathbf{on}(3, table), \\ \mathbf{goal}(1, 0), \mathbf{goal}(2, 1), \mathbf{goal}(3, 2) \end{array} \right\},$$

for which $\mathbf{incomplete}(0, 2)$ holds. This is a valid goal tower since block 4 does not occur in the goal description, i.e. does not need to be moved. Considering $s_3 \cup \{\mathbf{goal}(4, 3)\}$ instead, there exists no goal tower.

CARCASS Encoding. There are six abstract states, covering the following cases:

- $\hat{s}_0$: an incomplete tower exists that is clear and the next missing block is clear;
- $\hat{s}_1$: an incomplete tower exists but has other misplaced blocks on top;
- $\hat{s}_2$: an incomplete tower that is clear exists, but the next missing block is not clear;
- $\hat{s}_3$: no incomplete tower exists but the first block to construct one is not clear;
- $\hat{s}_4$: no incomplete tower exists and the first block to construct one is clear; and
- $\hat{s}_\infty$: all other states, i.e. goal states.

To illustrate our encoding, we discuss $\hat{s}_0$ and $\hat{s}_1$ in detail, including their admissible abstract actions. The abstract state $\hat{s}_0$ is encoded as

$$P_{\hat{s}_0} = \left\{ \begin{array}{l} \mathbf{sIdx}(r0, 0). \\ \mathbf{sCov}(r0, (T, N)) \leftarrow \mathbf{incomplete}(\_, T), \mathbf{cl}(T), \mathbf{goal}(N, T), \mathbf{cl}(N). \end{array} \right\}.$$

The next move towards a goal state is to place the next missing block N on top of the incomplete tower T, which is captured by the abstract action $\hat{a}_{0,0}$, encoded as

$$P_{\hat{s}_0, \hat{a}_{0,0}} = \left\{ \mathbf{aCov}(move(n, t), move(N, T)) \leftarrow \mathbf{sChoice}(r0), \mathbf{sCov}(r0, (T, N)). \right\}.$$

As a second abstract action, we define $\hat{a}_{i,\infty}$, as a catch-all, covering actions that are not covered by $\hat{a}_{0,0}$. It is encoded as

$$P_{\hat{s}_i, \hat{a}_{i,\infty}} = \left\{ \begin{array}{ll} \mathbf{aCov}(others, move(X, Y)) \leftarrow & \mathbf{cl}(X), \mathbf{cl}(Y), X! = Y, X! = table, \\ & not\ \mathbf{aCov}(move(\_, \_), move(X, Y)), \\ & not\ \mathbf{on}(X, Y). \end{array} \right\},$$

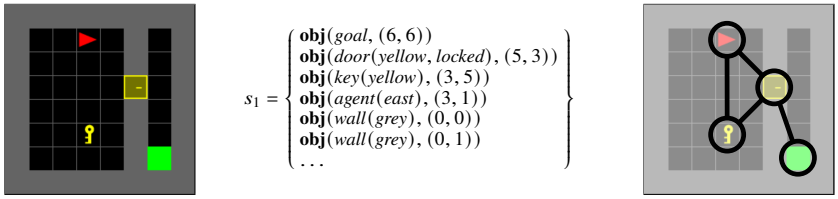


Fig. 2. Example Minigrid state from the Door Key 8×8 task environment. The left image shows a view of the state as it is usually presented. The center shows a relational representation of the same state. The right image shows a graph of the room layout, inferred using our ASP encoding.

such that it can be reused also for the other abstract states. Next, $\hat{s}_1$ is encoded as

$$P_{\hat{s}_1} = \left\{ \begin{array}{l} \textbf{sIdx}(r1, 1). \\ \textbf{sCov}(r1, (L, T, B)) \leftarrow \textbf{incomplete}(L, T), \mathit{not}\ \textbf{cl}(T), \textbf{above}(B, T), \textbf{cl}(B). \end{array} \right\}$$

with the admissible abstract actions $\hat{a}_{1,0}$ and $\hat{a}_{1,1}$:

$$P_{\hat{s}_1,\hat{a}_{1,0}} = \left\{ \begin{array}{l} \textbf{aCov}(move(b, table), move(B, table)) \leftarrow \textbf{sChoice}(r1), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{sCov}(r1, (\_, \_, B)). \end{array} \right\}$$

$$P_{\hat{s}_1,\hat{a}_{1,1}} = \left\{ \begin{array}{l} \textbf{aCov}(move(b, o), move(B, O)) \leftarrow \textbf{sChoice}(r1), \textbf{sCov}(r1, (L, T, B)), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{cl}(O), O \neq B, O \neq table. \end{array} \right\}$$

To advance $\hat{s}_1$ towards a goal state, all blocks on top of T need to be removed. The topmost block B can be placed either on the table ($\hat{a}_{1,0}$) or on top of some other free block ($\hat{a}_{1,1}$). All other actions are again covered by $\hat{a}_{i,\infty}$.

Analogy to GN1 and GN2 [33]. The abstract state $\hat{s}_0$ covers case (2) of the GN1 algorithm and moves covered by $\hat{a}_{0,0}$ are constructive moves. The abstract states $\hat{s}_1$ and $\hat{s}_2$ cover states that are deadlocked and relate to the two cases in the definition of the function δ_π.

5 Case Study: MiniGrid

Next we consider environments from the MiniGrid simulation library [4]. Solving such an environment typically requires the agent to navigate through a fixed layout of connected, rectangular rooms to reach the location of a goal tile as fast as possible, solving simple puzzles and avoiding hazards along the way.

Example 5 (Door Key Environment). In state s_1 (Fig 2), the agent and a key are located in one room and the goal tile in the other, separated by a locked door. The room layout changes between episodes, which makes the learning task more difficult.

We present the encoding of a state abstraction that is applicable to a variety of such environments. Background knowledge is used to infer a room layout from the raw state description, to build a graph based on the location of objects in that layout, and to compute the shortest path on that graph from the agent to the goal tile, representing an ordered set of subtasks that the agent needs to accomplish. Paths can be further constrained, for example to ensure that the key is visited before the door. The abstract state is constructed such that it contains only information relevant to the first node on the path. The full encoding of the abstraction is available in Appendix C.

MiniGrid as RMDP. Per default, a MiniGrid state is only partially observable and represented as a numeric matrix. We apply the *Fully Obs Wrapper*, such that the state description resembles a birds-eye view of the grid world, and use a custom relational state description with the following atoms: $\mathbf{obj}(O, (X, Y))$ asserts that object O is located at the grid coordinates (X, Y); $\mathbf{carries}(O)$ asserts that the agent carries object O; and **terminal** denotes the end of an episode. The actions are **right**, **left**, **forward**, **pickup**, **drop**, **toggle**, and **done**.

Background Knowledge. The key components of our abstraction, inferred using background knowledge, are represented with the following atoms: $\mathbf{oriented}(O)$ asserts the orientation of the agent; $\mathbf{fronting}((X, Y))$ asserts that (X, Y) is directly in front of the agent; $\mathbf{dangerous}((X, Y))$ denotes the existence of a dangerous object at (X, Y); $\mathbf{xdir}(X)$ and $\mathbf{ydir}(Y)$ assert the general horizontal and vertical direction, respectively, of the next object on the path in relation to the agent; $\mathbf{touching}(T)$ asserts that the next object T on the path is directly in front of the agent; and $\mathbf{ingap}(G)$ asserts that the agent is in a gap (e.g. between rooms).

Example 6. The computed path in s_1 (Fig. 2) is $(3, 1) - (3, 5) - (5, 3) - (6, 6)$. Based on the next node on that path, $(3, 5)$, the state is augmented with the facts $\mathbf{oriented}(east)$, $\mathbf{fronting}((4, 1))$, $\mathbf{xdir}(onAxis)$, $\mathbf{ydir}(south)$, $\mathbf{touching}(none)$, and $\mathbf{ingap}(none)$.

CARCASS Encoding. We define 757 abstract states (excluding $\hat{s}_\infty$). The first covers potentially dangerous situations and is encoded as

$$P_{\hat{s}_1} = \left\{ \begin{array}{l} \mathbf{sIdx}(danger, 1). \\ \mathbf{sCov}(danger, ()) \leftarrow \mathbf{fronting}(T), \mathbf{dangerous}(T). \end{array} \right\}.$$

The states $\hat{s}_2$–$\hat{s}_{757}$ are compactly encoded as

$$P_{\hat{s}_{2-757}} = \left\{ \begin{array}{l} \mathbf{label}((oriented(O), xdir(X), ydir(Y), touching(T), ingap(G))) \\ \quad \leftarrow \mathbf{oriented}(O), \mathbf{xdir}(X), \mathbf{ydir}(Y), \mathbf{touching}(T), \mathbf{ingap}(G). \\ \mathbf{sIdx}(S, 2) \leftarrow \mathbf{label}(S). \\ \mathbf{sCov}(S, ()) \leftarrow \mathbf{label}(S). \end{array} \right\},$$

and represent all possible combinations of arguments for $\mathbf{label}(T)$. The background knowledge is designed such that there can be at most one instance of **label** in any answer set, which means that the order restriction can be relaxed in favour of a more compact encoding, such that all states $\hat{s}_2$–$\hat{s}_{757}$ have the same index in the decision list. Actions can simply be stated as facts, e.g. $\mathbf{aCov}(left, left)$., but with the background knowledge it is also possible to prune the set of admissible actions by removing actions that have no effect. For example, $\mathbf{aCov}(pickup, pickup) \leftarrow \mathbf{fronting}(T), \mathbf{obj}(key(\_), T)$. ensures that the **pickup** action is admissible only if there is a key to be picked up.

6 Empirical Evaluation

We next investigate whether abstractions make large-scale RMDPs tractable by reducing both the space and sample requirements of the learning process. The convergence of

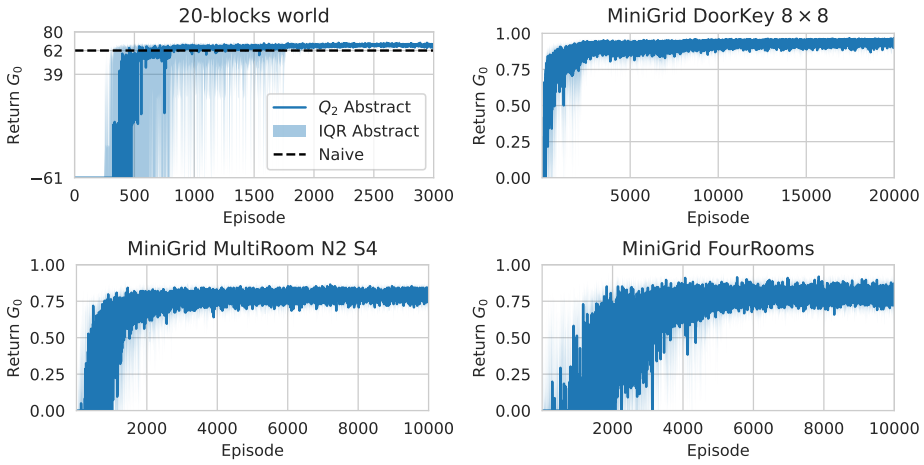


Fig. 3. The learning curves of abstract Q-learning in four task environments.

abstract Q-learning is however not guaranteed in general and there is also no guarantee that a learned abstract policy is optimal. Our experiments are thus designed to shed light on (i) the *stability of the learning process*, (ii) the *quality of the learned policy*, and (iii) the *differences in sample efficiency* between concrete and abstract Q-learning.

Experiment Setup. Both concrete and abstract Q-learning were tested in a 20-blocks world with the goal to stack all blocks in order, and in three MiniGrid environments. An ϵ-greedy behaviour policy was used for exploration; for details see Appendix D.

For every realised episode h_i, we measure the solution quality in terms of the *return* $\mathsf{G}_0(h_i)$ at time point zero. We further call an episode *successful* if it achieved a return above some baseline, $\mathsf{G}_0(h_i) > 0$ for MiniGrid, and $\mathsf{G}_0(h_i) \geq 62$ for the blocks world (i.e. the worst-case return of a naive unstack-stack strategy). For studying stability, we observe the return of episodes across the learning process. For studying sample efficiency, we observe the *mean return* $\frac{1}{n} \cdot \sum_{i=1}^{n} \mathsf{G}_0(h_i)$ and count the number of successful episodes over n realised episodes.

We repeated experiments 20 times. The collected data is summarised using the first quartile Q_1, the median Q_2, the third quartile Q_3, and the interquartile range IQR.

Results. Fig. 3 shows the learning curves of abstract Q-learning for the tested environments. For the 20-blocks world, the first positive median return is observed at episode 376 and there are only 10 instances after episode 2000 for which Q_1 drops below the baseline of 62, and none below 60. For the Door Key, Multi Room, and Four Rooms environments, the first positive median returns are observed after episodes 68, 133, and 287, respectively, all recorded median returns are positive after episodes 195, 1012, and 3137, respectively, and all recorded Q_1 returns are positive after episodes 2048, 1962, and 4649, respectively.

Table 2 shows (from left to right) the mean return over all realised episodes, the percentage of successful episodes over all realised episodes, and the percentage of

Table 2. Comparison of abstract and concrete Q-learning in four task environments.

Environment	Representation	Mean G_0 (all episodes)			# Successful (all episodes)			# Successful (last 500)		
		Q_1	Q_2	Q_3	Q_1	Q_2	Q_3	Q_1	Q_2	Q_3
20-blocks world	Abstract	18.64	43.86	54.07	0.40	0.76	0.85	0.93	0.97	0.98
	Concrete	-61.00	-61.00	-61.00	0.00	0.00	0.00	0.00	0.00	0.00
Door Key	Abstract	0.82	0.86	0.90	0.93	0.97	0.99	1.00	1.00	1.00
	Concrete	0.01	0.01	0.01	0.02	0.02	0.03	0.02	0.02	0.04
Four Rooms	Abstract	0.59	0.61	0.64	0.78	0.81	0.84	1.00	1.00	1.00
	Concrete	0.02	0.02	0.02	0.03	0.03	0.04	0.03	0.04	0.04
Multi Room	Abstract	0.68	0.69	0.70	0.91	0.91	0.92	1.00	1.00	1.00
	Concrete	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03

successful episodes over the last 500 episodes for abstract and concrete Q-learning in the four tested environments. Clear differences can be observed in all environments when comparing abstract and concrete Q-learning.

Discussion. Regarding *stability*, after initial learning, the returns for abstract Q-learning stay consistently high in all environments, with no visible fluctuations in the median or the interquartile range. Therefore, if stability issues are present, we conclude that the majority of episodes realised by abstract Q-learning are unaffected. Still, stability issues may be present in a minority (i.e., less than 25%) of experiment repetitions.

Regarding *quality*, the high success rates of episodes over the last 500 episodes suggest that episodes of acceptable quality can be learned by abstract Q-learning, for all environments and with high consistency. We can make no statements about the optimality of the learned policies. However, for the 20-blocks world, there are still episodes where the learned policy does not match the worst-case return of a naive unstack-stack policy. This may be caused by the exploration policy, by other factors (e.g., choice of α, more training needed), or by the abstraction itself.

For *sample efficiency*, we can conclude from the differences in the mean returns and the percentages of successful episodes over the entire learning processes that abstract Q-learning has higher sample efficiency than concrete Q-learning for the investigated environments. That is, abstract Q-learning reliably produces policies of acceptable quality *in a smaller number of episodes* as compared to concrete Q-learning.

7 Related Work

Research on relational reinforcement learning has long explored ways of representing state-action abstractions in first-order logic. Prominent examples include *logical Markov decision programs* (LOMDPs) [17], which describe abstract states as logical rules and transitions as probabilistic effects of actions, and *relational Q-learning* [22], which groups states and actions into logical templates that define when an abstract action can be applied. Other extensions include *probabilistic relational models* (PRMs) [13], which decompose value functions across object classes, and work on *stochastic policies*

for relational POMDPs [14], where policies are expressed as decision lists with probabilities. All of these approaches share with *CARCASS* [37] the goal of reducing the state explosion problem through logical abstraction, but they differ in their underlying mechanisms: LOMDPs match concrete states against general logical rules, *RQ*-learning defines partitions of the state-action space through conjunctions of relations, and PRMs assume additive decompositions over objects. CARCASS instead represents abstractions as logical rules with background knowledge, implemented in Prolog using SLDNF resolution. Our work continues this line but replaces Prolog with a fully declarative ASP encoding, which allows richer forms of reasoning, including nonmonotonic inference and optimisation, within the abstraction process.

Several works also combine *ASP and reinforcement learning*, though in very different ways. For instance, relational meta-policies have been encoded in ASP for hierarchical option selection [21], and other approaches use ASP to restrict exploration by pruning infeasible actions [9,5]. In robotics, ASP has been employed for reasoning about incomplete knowledge, planning, or diagnosis [41,44,34,23]. ASP has also been used to encode entire relational MDPs, including states and value functions [28]. However, none of these works proposed a general ASP-based representation of *full state-action abstractions* in the style of CARCASS, which is the focus of our contribution.

Beyond logic-based approaches, *statistical methods* have been widely used in relational RL, including regression trees [6,7], kernel methods [10], and more recently deep relational models [43,16]. Relational planning languages such as *RDDL* [29] and *PPDDL* [42] also enable compact, symbolic formulations of RMDPs, which can be solved without grounding into all concrete states, for instance using lifted value-function approximation techniques [18,30].

8 Conclusion

We investigated ASP as a representation for state-action pair abstractions in relational MDPs, building on the CARCASS framework. We presented a general encoding method and evaluated it on Blocks World and MiniGrid tasks. In both domains, abstract representations enabled Q-learning to learn high-quality policies consistently with fewer samples than concrete representations, showing its potential as a knowledge-driven framework for relational RL abstractions that supports elaboration-tolerant policy learning. ASP-based abstractions support generalisation, action restrictions, and problem decomposition by allowing declarative encoding of domain knowledge. They also support nonmonotonic reasoning, optimisation, and reasoning with incomplete knowledge, providing a flexible alternative to Prolog-based CARCASS abstractions.

Future work includes automatic generation and refinement of ASP abstractions [31], leveraging nondeterminism and optimisation to guide exploration, and testing transfer of abstract policies to larger or previously unseen tasks. Improving computational efficiency via multi-shot solving and assumptions [12] is also a key avenue for scaling to more complex environments. Additional directions include exploring elaboration-tolerant design patterns, integrating ASP abstractions with function approximation methods, and evaluating the benefits of symbolic priors for RL in richer relational settings.

Acknowledgments. This work has been supported by funding from the Knowledge Foundation (KK-stiftelsen) within the synergy project *Efficient and Trustworthy Industrial AI* (ETIAI).

References

1. Bankosegger, R.: ASP-Based Abstractions for Reinforcement Learning. Master's thesis, Technische Universität Wien (2025), https://doi.org/10.34726/hss.2025.90825
2. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011). https://doi.org/10.1145/2043174.2043195, https://doi.org/10.1145/2043174.2043195
3. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., Schaub, T.: ASP-Core-2 input language format. Theory Pract. Log. Program. **20**(2), 294–309 (2020). https://doi.org/10.1017/S1471068419000450, https://doi.org/10.1017/S1471068419000450
4. Chevalier-Boisvert, M., Dai, B., Towers, M., de Lazcano, R., Willems, L., Lahlou, S., Pal, S., Castro, P.S., Terry, J.: Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. CoRR **abs/2306.13831** (2023). https://doi.org/10.48550/ARXIV.2306.13831, https://doi.org/10.48550/arXiv.2306.13831
5. Datler, E.: Balancing learning and planning: A case study on combining reinforcement learning with answer set programming in the blocks world. Bachelor's Thesis, Technische Universität Wien (2020)
6. Dzeroski, S., De Raedt, L., Blockeel, H.: Relational reinforcement learning. In: Shavlik, J.W. (ed.) Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998. pp. 136–143. Morgan Kaufmann (1998)
7. Džeroski, S., Raedt, L.D., Driessens, K.: Relational reinforcement learning. Mach. Learn. **43**(1/2), 7–52 (2001). https://doi.org/10.1023/A:1007694015589, https://doi.org/10.1023/A:1007694015589
8. Eiter, T., Ianni, G., Krennwallner, T.: Answer set programming: A primer. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M., Schmidt, R.A. (eds.) Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures. Lecture Notes in Computer Science, vol. 5689, pp. 40–110. Springer (2009). https://doi.org/10.1007/978-3-642-03754-2_2, https://doi.org/10.1007/978-3-642-03754-2_2
9. Ferreira, L.A., Bianchi, R.A.C., Santos, P.E., de Mántaras, R.L.: Answer set programming for non-stationary markov decision processes. Appl. Intell. **47**(4), 993–1007 (2017). https://doi.org/10.1007/S10489-017-0988-Y, https://doi.org/10.1007/s10489-017-0988-y
10. Gärtner, T., Driessens, K., Ramon, J.: Graph kernels and gaussian processes for relational reinforcement learning. In: Horváth, T. (ed.) Inductive Logic Programming: 13th International Conference, ILP 2003, Szeged, Hungary, September 29-October 1, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2835, pp. 146–163. Springer (2003). https://doi.org/10.1007/978-3-540-39917-9_11, https://doi.org/10.1007/978-3-540-39917-9_11
11. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012). https://doi.org/10.2200/S00457ED1V01Y201211AIM019, https://doi.org/10.2200/S00457ED1V01Y201211AIM019
12. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory Pract. Log. Program. **19**(1), 27–82 (2019). https://doi.org/10.1017/S1471068418000054, https://doi.org/10.1017/S1471068418000054
13. Guestrin, C.E.: Planning under uncertainty in complex structured environments. Stanford University (2003)
14. Itoh, H.: Towards learning to learn and plan by relational reinforcement learning. In: Proceedings of the Workshop on Relational Reinforcement Learning at the Twenty-First International Conference on Machine Learning (RRL at ICML-2004). pp. 34–39 (2004)

15. Jaakkola, T.S., Jordan, M.I., Singh, S.P.: On the convergence of stochastic iterative dynamic programming algorithms. Neural Comput. **6**(6), 1185–1201 (1994). https://doi.org/10.1162/NECO.1994.6.6.1185, https://doi.org/10.1162/neco.1994.6.6.1185
16. Janisch, J., Pevný, T., Lisý, V.: Symbolic relational deep reinforcement learning based on graph neural networks. CoRR **abs/2009.12462** (2020), https://arxiv.org/abs/2009.12462
17. Kersting, K., De Raedt, L.: Logical markov decision programs and the convergence of logical td(lambda). In: Camacho, R., King, R.D., Srinivasan, A. (eds.) Inductive Logic Programming, 14th International Conference, ILP 2004, Porto, Portugal, September 6-8, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3194, pp. 180–197. Springer (2004). https://doi.org/10.1007/978-3-540-30109-7_16, https://doi.org/10.1007/978-3-540-30109-7_16
18. Kersting, K., van Otterlo, M., De Raedt, L.: Bellman goes relational. In: Brodley, C.E. (ed.) Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004. ACM International Conference Proceeding Series, vol. 69. ACM (2004). https://doi.org/10.1145/1015330.1015401, https://doi.org/10.1145/1015330.1015401
19. Li, L., Walsh, T.J., Littman, M.L.: Towards a unified theory of state abstraction for mdps. In: International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006 (2006), http://anytime.cs.umass.edu/aimath06/proceedings/P21.pdf
20. Lifschitz, V.: Answer Set Programming. Springer (2019). https://doi.org/10.1007/978-3-030-24658-7, https://doi.org/10.1007/978-3-030-24658-7
21. Mitchener, L., Tuckey, D., Crosby, M., Russo, A.: Detect, understand, act: A neuro-symbolic hierarchical reinforcement learning framework. Mach. Learn. **111**(4), 1523–1549 (2022). https://doi.org/10.1007/S10994-022-06142-7, https://doi.org/10.1007/s10994-022-06142-7
22. Morales, E.F.: Scaling up reinforcement learning with a relational representation. In: Proceedings of the Workshop on Adaptability in Multi-Agent Systems at AORC 2003, Sydney, Australia (2003)
23. Mota, T., Sridharan, M., Leonardis, A.: Integrated commonsense reasoning and deep learning for transparent decision making in robotics. SN Comput. Sci. **2**(4), 242 (2021). https://doi.org/10.1007/S42979-021-00573-0, https://doi.org/10.1007/s42979-021-00573-0
24. Nienhuys-Cheng, S., de Wolf, R.: Foundations of Inductive Logic Programming, Lecture Notes in Computer Science, vol. 1228. Springer (1997). https://doi.org/10.1007/3-540-62927-0, https://doi.org/10.1007/3-540-62927-0
25. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887, https://doi.org/10.1002/9780470316887
26. Ravindran, B.: An Algebraic Approach to Abstraction in Reinforcement Learning. Ph.D. thesis, University of Massachusetts Amherst (2 2004)
27. Ravindran, B., Barto, A.G.: SMDP homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In: Gottlob, G., Walsh, T. (eds.) IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003. pp. 1011–1018. Morgan Kaufmann (2003), http://ijcai.org/Proceedings/03/Papers/145.pdf
28. Saad, E.: Bridging the gap between reinforcement learning and knowledge representation: A logical off- and on-policy framework. In: Liu, W. (ed.) Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 11th European Conference, ECSQARU 2011, Belfast, UK, June 29-July 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6717, pp. 472–484. Springer (2011). https://doi.org/10.1007/978-3-642-22152-1_40, https://doi.org/10.1007/978-3-642-22152-1_40
29. Sanner, S.: Relational dynamic influence diagram language (RDDL): Language description (2010), https://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf

30. Sanner, S., Boutilier, C.: Practical solution techniques for first-order mdps. Artif. Intell. **173**(5-6), 748–788 (2009). https://doi.org/10.1016/J.ARTINT.2008.11.003, https://doi.org/10.1016/j.artint.2008.11.003
31. Saribatur, Z.G., Eiter, T., Schüller, P.: Abstraction for non-ground answer set programs. Artif. Intell. **300**, 103563 (2021). https://doi.org/10.1016/J.ARTINT.2021.103563, https://doi.org/10.1016/j.artint.2021.103563
32. Singh, S., Jaakkola, T.S., Jordan, M.I.: Reinforcement learning with soft state aggregation. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]. pp. 361–368. MIT Press (1994), https://proceedings.neurips.cc/paper_files/paper/1994/hash/287e03db1d99e0ec2edb90d079e142f3-Abstract.html
33. Slaney, J.K., Thiébaux, S.: Blocks world revisited. Artif. Intell. **125**(1-2), 119–153 (2001). https://doi.org/10.1016/S0004-3702(00)00079-5, https://doi.org/10.1016/S0004-3702(00)00079-5
34. Sridharan, M., Meadows, B.: Knowledge representation and interactive learning of domain knowledge for human-robot interaction. Advances in Cognitive Systems **7**, 77–96 (2018), http://www.cogsys.org/journal/volume7/
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning - An Introduction, 2nd Edition. MIT Press (2018), http://www.incompleteideas.net/book/the-book-2nd.html
36. Tsitsiklis, J.N.: Asynchronous stochastic approximation and q-learning. Mach. Learn. **16**(3), 185–202 (1994). https://doi.org/10.1007/BF00993306, https://doi.org/10.1007/BF00993306
37. van Otterlo, M.: Reinforcement learning for relational MDPs. In: Nowé, A., Lenaerts, T., Steenhaut, K. (eds.) Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands (BeNeLearn 2004), January 8-9, 2004. pp. 138–145 (2004), https://web.archive.org/web/20040715063749/http://benelearn04.vub.ac.be/BENELEARN.pdf
38. van Otterlo, M.: A Survey of Reinforcement Learning in Relational Domains. No. 05-31 in CTIT Technical Report Series (2005), https://research.utwente.nl/en/publications/a-survey-of-reinforcement-learning-in-relational-domains
39. van Otterlo, M.: The logic of adaptive behavior. Ph.D. thesis, University of Twente, Enschede, the Netherlands (5 2008)
40. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**(3), 279–292 (1992). https://doi.org/10.1007/BF00992698, https://doi.org/10.1007/BF00992698
41. Yang, F., Khandelwal, P., Leonetti, M., Stone, P.: Planning in answer set programming while learning action costs for mobile robots. In: 2014 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 24-26, 2014. AAAI Press (2014), http://www.aaai.org/ocs/index.php/SSS/SSS14/paper/view/7727
42. Younes, H.L.S., Littman, M.L., Weissman, D., Asmuth, J.: The first probabilistic track of the international planning competition. J. Artif. Intell. Res. **24**, 851–887 (2005). https://doi.org/10.1613/JAIR.1880, https://doi.org/10.1613/jair.1880
43. Zambaldi, V.F., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D.P., Lillicrap, T.P., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M.M., Vinyals, O., Battaglia, P.W.: Deep reinforcement learning with relational inductive biases. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=HkxaFoC9KQ
44. Zhang, S., Sridharan, M., Wyatt, J.L.: Mixed logical inference and probabilistic planning for robots in unreliable worlds. IEEE Trans. Robotics **31**(3), 699–713 (2015). https://doi.org/10.1109/TRO.2015.2422531, https://doi.org/10.1109/TRO.2015.2422531

A Proof of Proposition 1

Proof (⇒). Assume that $P_s \vdash_{\text{SLDNF}} \hat{s}_i\theta$. The SLDNF-refutation of $P_s \cup \{\leftarrow \hat{s}_i\theta.\}$ is a sequence of derivation steps with goal clauses

$$(\leftarrow l_1\theta, \ldots, l_{k-1}\theta, l_k\theta.), (\leftarrow l_1\theta, \ldots, l_{k-1}\theta.), \ldots (\leftarrow l_1\theta.), (\square.).$$

At every derivation step, a literal $l_j\theta$ is deleted from the body (assuming an arbitrary order without loss of generality).

- If $l_j\theta = \mathbf{p}(\bar{t})$ is positive, then $(\leftarrow l_1\theta, \ldots, l_{j-1}\theta.)$ is a binary resolvent of $(\leftarrow l_1\theta, \ldots, l_{j-1}\theta, \mathbf{p}(\bar{t}).)$ and the fact $(\mathbf{p}(\bar{t}).) \in P_s$. It follows, since $P_s \subset P$, that $\mathbf{p}(\bar{t}) = l_j\theta$ is true in I.
- If $l_j\theta = \mathit{not}\ \mathbf{p}(\bar{t})$ is negative, then there exists a finitely failed subsidiary derivation tree with $(\leftarrow \mathbf{p}(\bar{t}).)$ as root and no children, which can only mean that $(\mathbf{p}(\bar{t}).) \notin P_s$, and thus, as there are no rules in P with $\mathbf{p}$ in their head, that $\mathit{not}\ \mathbf{p}(\bar{t}) = l_j\theta$ is true in I.

Now observe that there exists a ground rule $r = (\mathbf{sCov}(\lambda(\hat{s}_i), (V_1\theta, \ldots, V_m\theta)) \leftarrow l_1\theta, \ldots, l_k\theta.) \in \mathit{Gnd}(P)$. This is clear since a non-ground rule of this form is in $P_{\hat{s}_i} \subset P$ and because $\hat{s}_i\theta$ is a ground instance of $\hat{s}_i$, the body of r. Since $l_j\theta$ is true in I for all j, it follows that $l_1\theta, \ldots, l_k\theta = \mathit{body}(r)$ is true in I and therefore that $\mathbf{sCov}(\lambda(\hat{s}_i), (V_1\theta, \ldots, V_m\theta))$ is true in I, i.e. $\mathbf{sCov}(\lambda(\hat{s}_i), (V_1\theta, \ldots, V_m\theta)) \in I$.

Proof (⇐). Assume that $\mathbf{sCov}(\lambda(\hat{s}_i), (V_1\theta, \ldots, V_m\theta)) \in I$. This atom must be supported by some rule in $r \in \mathit{Gnd}(P)$ with $\mathit{body}(r)$ true in I [8, Theorem 6], which can only be of the form $r = (\mathbf{sCov}(\lambda(\hat{s}_i), (V_1\theta, \ldots, V_m\theta)) \leftarrow l_1\theta, \ldots, l_k\theta.)$. We can inductively construct an SLDNF-refutation $P_s \cup \{\leftarrow \hat{s}_i\theta.\}$ with the following key insight:

- If some $l_j\theta = \mathbf{p}(\bar{t})$ is positive, then it follows that $\mathbf{p}(\bar{t})$ is true in I, since $\hat{s}_i = \mathit{body}(r)$ is true in I. Thus, $\mathbf{p}(\bar{t})$ must be supported, which can only be due to the fact $(\mathbf{p}(\bar{t}).) \in P_s$. Then, $(\leftarrow l_1\theta, \ldots, l_{j-1}\theta.)$ is a binary resolvent of $(\leftarrow l_1\theta, \ldots, l_{j-1}\theta, l_j\theta.)$ and of $(\mathbf{p}(\bar{t}).)$.
- If $l_j\theta = \mathit{not}\ \mathbf{p}(\bar{t})$ is negative, then it follows that $(\mathbf{p}(\bar{t}).) \notin I$ since $l_j\theta$ is true in I, and thus $(\mathbf{p}(\bar{t}).) \notin P_s$. So, the derivation $(\leftarrow \mathbf{p}(\bar{t}).)$ fails finitely and $l_j\theta$ can be deleted from the goal clause $(\leftarrow l_1\theta, \ldots, l_{j-1}\theta, l_j\theta.)$.

B ASP Encoding: Blocks World CARCASS

Domain Knowledge

```
cl(A) :- on(A,_), not on(_,A).
cl(A) :- table = A.
above(A,B) :- on(A,B).
above(A,B) :- on(A,C), above(C,B).
goalRelevant(A) :- goal(A,_).
goalRelevant(A) :- goal(_, A).
finalBasis(X) :- goal(_,X), not goal(X,_),
  0 = #count { B : above(X,B), goalRelevant(B) }.
final(X, X) :- finalBasis(X).
final(X,B1) :- final(X,B2), on(B1,B2), goal(B1,B2).
incomplete(X,T) :- final(X,T), goal(B,T), not on(B,T).
```

CARCASS Encoding

```
1 { sChoice(R) : sCov(R,_) } 1.
:- sChoice(R), sIdx(R,I1), sIdx(R2,I2), sCov(R2,_), I2<I1.

sIdx(r0,0).
sCov(r0,(Top,Next)) :-
  incomplete(_,Top), cl(Top), goal(Next,Top), cl(Next).
aCov(move(next,top), move(Next,Top)) :- sChoice(r0), sCov(r0,(
    Top,Next)).

sIdx(r1,1).
sCov(r1,(Basis,Top,BadTop)) :-
  incomplete(Basis,Top), not cl(Top),
  above(BadTop,Top), cl(BadTop).
aCov(move(badTop,table),move(BadTop,table)) :-
  sChoice(r1), sCov(r1,(_,_,BadTop)).
aCov(move(badTop,other),move(BadTop,Other)) :-
  sChoice(r1), sCov(r1,(Basis,Top,BadTop)),
  cl(Other), Other!=BadTop, Other!=table.

sIdx(r2,2).
sCov(r2,(Top,BadTop)) :-
  incomplete(Basis,Top), cl(Top), goal(Next,Top),
  not cl(Next), above(BadTop,Next), cl(BadTop).
aCov(move(badTop,table), move(BadTop,table)) :-
  sChoice(r2), sCov(r2,(_,BadTop)).
aCov(move(badTop,otherTowerTop), move(BadTop,Other)) :-
  sChoice(r2), sCov(r2,(Top,BadTop)),
  cl(Other), Other!=Top, Other!=BadTop, Other!=table.

sIdx(r3,3).
sCov(r3,(BadTop)) :-
  not finalBasis(_), goal(_,Basis), not goal(Basis,_),
  not cl(Basis), above(BadTop,Basis), cl(BadTop).
```

```
aCov(move(badTop,table), move(BadTop,table)) :-
  sChoice(r3), sCov(r3,(BadTop)).
aCov(move(badTop,otherTowerTop), move(BadTop,Other)) :-
  sChoice(r3), sCov(r3,(BadTop)), cl(Other),
  Other!=BadTop, Other!=table.

sIdx(r4,4).
sCov(r4,(Basis)) :-
  not finalBasis(_), goal(_,Basis),
  not goal(Basis,_), cl(Basis).
aCov(move(basis,table), move(Basis,table)) :-
  sChoice(r4), sCov(r4,(Basis)).

sIdx(true,#sup).
sCov(true,()).
aCov(others, move(X,Y)) :-
  cl(X), cl(Y), X!=Y, X!=table,
  not aCov(move(_,_),move(X,Y)), not on(X,Y).
```

C ASP Encoding: MiniGrid CARCASS

Domain Knowledge

```
blocked(XY) :- obj(wall(_),XY).
blocked(XY) :- obj(lava,XY).
gap((X,Y),horizontal) :-
  not blocked((X,Y)), blocked((X,Y-1)), blocked((X,Y+1)).
gap((X,Y),vertical) :-
  not blocked((X,Y)), blocked((X-1,Y)), blocked((X+1,Y)).

alley((X,Y)) :- gap((X,Y),horizontal), gap((X+1,Y), horizontal).
alley((X,Y)) :- gap((X,Y),horizontal), gap((X-1,Y), horizontal).
alley((X,Y)) :- gap((X,Y),vertical), gap((X,Y+1), vertical).
alley((X,Y)) :- gap((X,Y),vertical), gap((X,Y-1), vertical).

pcp(U) :- blocked(U).
pcp(U) :- gap(U,_), not alley(U).

rtl((X,Y)) :- pcp((X,Y)), pcp((X+1,Y)), pcp((X,Y+1)).
rbl((X,Y)) :- pcp((X,Y)), pcp((X+1,Y)), pcp((X,Y-1)).
rtr((X,Y)) :- pcp((X,Y)), pcp((X-1,Y)), pcp((X,Y+1)).
rbr((X,Y)) :- pcp((X,Y)), pcp((X-1,Y)), pcp((X,Y-1)).

rect(((X1,Y1), (X2,Y2))) :-
  rtl((X1,Y1)), rbl((X1,Y2)),
  rtr((X2,Y1)), rbr((X2,Y2)),
  X2-X1>0, Y2-Y1>0.
contains(((X11,Y11),(X12,Y12)), ((X21,Y21),(X22,Y22))) :-
  rect(((X11,Y11),(X12,Y12))), rect(((X21,Y21),(X22,Y22))),
  X11<=X21, Y11<=Y21, X12>=X22, Y12>=Y22,
  ((X11,Y11),(X12,Y12))!=((X21,Y21),(X22,Y22)).
room(R) :- rect(R), not contains(R,_).

v(U) :- obj(door(_,_),U).
v(U) :- obj(goal,U).
v(U) :- obj(agent(_),U).
v(U) :- obj(key(_),U).
v(U) :- obj(ball(C),U), not C=blue.
v(U) :- gap(U,_), not alley(U).

inRoom((X,Y),((X1,Y1),(X2,Y2))) :-
  v((X,Y)), room(((X1,Y1),(X2,Y2))),
  X1<=X, Y1<=Y, X2>=X,Y2>=Y.
e(U,V) :- v(U), v(V), inRoom(U,R), inRoom(V,R), U!=V.

p(0,U) :- obj(agent(_),U).
0 { p(T+1,V) : e(U,V) } 1 :- p(T,U), T<#count{ X : v(X) }.
:- obj(goal,_), 0 = #count{ U : p(_,U), obj(goal,U) }, not
    terminal.
:~ p(T,_). [1@2,T]
```

```
:~ p(1,(X1,Y1)), obj(agent(_),(X2,Y2)), D=|X2-X1|+|Y2-Y1|. [D@1]

needsKeyBefore(C,T) :- p(T,U), obj(door(C,locked),U), not
    carries(key(C)).
keyOnPath(C,T) :- p(T,U), obj(key(C),U).
:- 0=#count{ T : keyOnPath(C,T), T<T1}, needsKeyBefore(C,T1).

nextOnPath(V) :- p(1,V).

adj((X+1,Y)) :- obj(agent(_),(X,Y)).
adj((X-1,Y)) :- obj(agent(_),(X,Y)).
adj((X,Y+1)) :- obj(agent(_),(X,Y)).
adj((X,Y-1)) :- obj(agent(_),(X,Y)).

fronting((X,Y-1)) :- obj(agent(north),(X,Y)).
fronting((X,Y+1)) :- obj(agent(south),(X,Y)).
fronting((X+1,Y)) :- obj(agent(east),(X,Y)).
fronting((X-1,Y)) :- obj(agent(west),(X,Y)).

dangerous(XY) :- obj(ball(blue),XY).
dangerous(XY) :- obj(lava,XY).

oriented(O) :- obj(agent(O),_).

xdir(west) :- obj(agent(_),(AX,AY)), nextOnPath((GX,GY)), AX>GX.
xdir(east) :- obj(agent(_),(AX,AY)), nextOnPath((GX,GY)), AX<GX.
xdir(on_axis) :- obj(agent(_),(AX,AY)), nextOnPath((GX,GY)), AX=
    GX.

ydir(north) :- obj(agent(_),(AX,AY)), nextOnPath((GX,GY)), AY>GY
    .
ydir(south) :- obj(agent(_),(AX,AY)), nextOnPath((GX,GY)), AY<GY
    .
ydir(on_axis) :- obj(agent(_),(AX,AY)), nextOnPath((GX,GY)), AY=
    GY.

touching(goal) :- nextOnPath(G), adj(G), obj(goal,G).
touching(key) :- nextOnPath(G), adj(G), obj(key(_),G).
touching(door(S)) :- nextOnPath(G), adj(G), obj(door(_, S),G).
touching(gap) :- nextOnPath(G), adj(G), gap(G,_), not alley(G),
  not touching(goal), not touching(door(_)), not touching(key).
touching(none) :- not touching(goal), not touching(door(_)),
  not touching(gap), not touching(key).

in_gap(D) :- obj(agent(_),XY), gap(XY,D), not alley(XY).
in_gap(none) :- not in_gap(horizontal), not in_gap(vertical).
```

CARCASS Encoding

```
1 { sChoice(R) : sCov(R,_) } 1.
:- sChoice(R1), sCov(R2,_), sIdx(R1,I1), sIdx(R2,I2), I2<I1.
```

```
sIdx(danger,1).
sCov(danger,()) :- fronting(T), dangerous(T).

label((oriented(O),xdir(X),ydir(Y),touching(T),in_gap(G))) :-
  oriented(O), xdir(X), ydir(Y), touching(T), in_gap(G).
sIdx(S,2) :- label(S).
sCov(S,()) :- label(S).

sIdx(true,#sup).
sCov(true,()).

aCov(left,left).
aCov(right,right).

forwardMoveBlocked :- fronting(XY), obj(wall(_),XY).
forwardMoveBlocked :- fronting(XY), obj(door(_,closed),XY).
forwardMoveBlocked :- fronting(XY), obj(door(_,locked),XY).
forwardMoveBlocked :- fronting(XY), obj(key(_),XY).
aCov(forward,forward) :- not forwardMoveBlocked.

aCov(pickup,pickup) :- fronting(XY), obj(key(_),XY).
aCov(toggle,toggle) :- fronting(XY), obj(door(_,_),XY).
aCov(drop,drop) :- carries(_), not forwardMoveBlocked.
```

D Experiment Setup

Implementation details. Algorithm 1.1 was used as implementation of abstract Q-learning and Algorithm 1.2 to compute the abstraction function. Note that it is possible during the learning process for one abstract state to have different sets of admissible abstract actions in different concrete states. In our implementation, this is handled by adding the set of admissible actions to the state description, forming a refined abstract state $\hat{\mathsf{X}}_t = (\hat{\mathsf{S}}_t, \hat{A}'_{\hat{\mathsf{S}}_t})$.

The Q-learning algorithm from Sutton and Barto [35, Chapter 6.5] was used on the concrete representations. The algorithms were implemented in *Python 3*[3], and *clingo 5*[4]. Our implementation is available online.[5] The experiments were run on a desktop PC with one *AMD Ryzen 5 Pro 4650G* CPU and 16GB memory.

Behaviour Policy. The ϵ-greedy policy [35, p. 101] is defined as

$$\pi_{\mathsf{Q},\epsilon}(s,a) \doteq \begin{cases} 1-\epsilon+\dfrac{\epsilon}{|A_s|} & \text{if } a = \arg\max_{a'}\{\mathsf{Q}(s,a')\} \text{ and} \\ \dfrac{\epsilon}{|A_s|} & \text{otherwise.} \end{cases}$$

20-Blocks World. The exact goal description used is $\mathbf{goal}(0, \mathit{table})$, $\mathbf{goal}(1,0)$, $\ldots$, $\mathbf{goal}(19,18)$. Initial states are drawn from a random uniform distribution [33, p. 126]. The parameters were set to $\alpha = 0.01$, $\epsilon = 0.05$, and $\hat{\mathsf{Q}}$ was initialized to -1 for all non-terminal states. For every episode, a time limit of $3n+1$ (where $n = 20$) was used. For every repetition of the experiment, the learning process was stopped after realising the first 3000 episodes.

MiniGrid Door Key 8 × 8. The exact environment used is *MiniGrid-DoorKey-8x8-v0*. The parameters were set to $\alpha = 0.00001$, $\epsilon = 0.2$, and $\hat{\mathsf{Q}}$ was initialized to 0.3 for all non-terminal states. For every episode, the standard time limit of 640 steps was used. For every repetition of the experiment, the learning process was stopped after realising the first 20.000 episodes.

MiniGrid Four Rooms. The exact environment used is *MiniGrid-FourRooms-v0*. The parameters were set to $\alpha = 0.001$, $\epsilon = 0.2$, and $\hat{\mathsf{Q}}$ was initialized to 0.3 for all non-terminal states. For every episode, the standard time limit of 100 steps was used. For every repetition of the experiment, the learning process was stopped after realising the first 10.000 episodes.

MiniGrid Multi Room N2 S4. The exact environment used is *MiniGrid-MultiRoom-N2-S4-v0*. The parameters were set to $\alpha = 0.001$, $\epsilon = 0.2$, and $\hat{\mathsf{Q}}$ was initialized to 0.3 for

[3] https://www.python.org/
[4] https://potassco.org/clingo/
[5] https://github.com/rbankosegger/RLASP-core.

all non-terminal states. For every episode, the standard time limit of 40 steps was used. For every repetition of the experiment, the learning process was stopped after realising the first 10.000 episodes.