TPLP **26** (1-2): 46–63, 2026. © The Author(s), 2026. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

doi:10.1017/S1471068426100544 First published online 8 July 2026

*Answer-Set-Programming-Based Abstractions for Reinforcement Learning**

RAFAEL BANKOSEGGER
Siemens AG Österreich, Austria and TU Wien, Austria
(*e-mail:* rafael-philipp.bankosegger@siemens.com)

THOMAS EITER
Institute of Logic and Computation, Technische Universitat Wien, Austria
(*e-mail:* eiter@kr.tuwien.ac.at)

JOHANNES OETSCH
Department of Computing, Jönköping University, Sweden
(*e-mail:* johannes.oetsch@ju.se)

submitted 29 May 2026 UTC; revised 29 May 2026 UTC; accepted 1 June 2026 UTC

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 Prologue, CARCASS leverages domain knowledge to create powerful abstractions. We explore Answer-Set Programming (ASP), which is a rich and, contrary to Prologue, fully declarative modelling language, to realise CARCASS abstractions. 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 (our implementation is available at https://github.com/rbankosegger/RLASP-core. Further material (data, encodings, extended documentation) can be found here: https://www.bankosegger.at/iclp26/.).

Keywords: Relational Reinforcement Learning, ASP, state-action-space abstractions

1 Introduction

Reinforcement Learning (RL) (Sutton and Barto 2018) has emerged as a central paradigm in AI, providing a principled framework for agents to learn behaviour through interaction

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

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

with an environment. By formalising sequential decision-making as a Markov Decision Process (MDP), RL enables agents to improve their behaviour by trial and error, guided by the maximisation of cumulative reward. Although this paradigm has produced impressive results with significant practical applications, a challenge that remains is that 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.

Approximations, for example, Deep Q-Learning (Mnih *et al*. 2015), aim at learning compact representations of expected long-term rewards, typically using deep neural networks, thereby enabling generalisation across large or continuous state spaces. In practice, however, this often requires large amounts of training data, and learning can be unstable (Sutton and Barto 2018, Chapter 11.3).

Relational Reinforcement Learning (RRL) addresses some of these limitations by using first-order logic to introduce powerful abstractions for reasoning about complex problem domains. The key idea is to extend representations to a first-order setting, allowing environments to be described naturally in terms of objects and their relations (van Otterlo 2005). A major advantage of this approach is that it supports generalisation across similar states and can even transfer learned knowledge to related tasks.

One particular approach to RRL, rooted in logic programming, is the CARCASS framework (short for *Compact Abstraction using Relational Conjunctions for Aggregation of State-action Spaces*) introduced by Martijn van Otterlo (2004). 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 $State \rightarrow \{\text{Action}_1, \ldots, \text{Action}_n\}$, each capturing a situation described by the state and the available actions. For example, the rule

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

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, for example, Q-learning (Watkins and Dayan 1992) 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 realised in Prologue, making use of SLDNF-resolution to handle first-order reasoning over states and actions (van Otterlo 2009).

Building on this, we explore Answer-Set Programming (ASP) (Brewka *et al*. 2011) as a rich and fully declarative modelling language for revisiting CARCASS.[1] ASP is a logic-based knowledge representation and reasoning framework, featuring a concise yet highly expressive modelling language. Unlike Prologue, the meaning of an ASP programme is independent of the ordering of rules and of literals in rule bodies. It supports nonmonotonic inference, reasoning under incomplete knowledge, expressing preferences

[1] Pure Prologue is declarative in principle, but in practice many implementations rely on procedural features such as negation as failure, cuts, or the ordering of rules to achieve efficiency.

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

and optimisation, as well as modelling actions and change, making it particularly suitable for describing state abstractions in RRL.

Reimplementing CARCASS in ASP highlights several practical advantages. Admissible states, transitions, and optimisation criteria can be specified declaratively at a high level, without explicit procedural search strategies. Defaults and integrity constraints are supported naturally, allowing concise and robust encodings that would otherwise require more algorithmic, control-intensive implementations in Prologue. This also enables direct representation of complex reasoning patterns, including partial observability, nondeterminism, and preference-based selection, with domain constraints expressed directly in the rules.

In this work, we introduce a general method for encoding abstractions in ASP, which we evaluate using two case studies. The first considers Blocks World, a classical planning problem that involves stacking blocks to reach a desired configuration, and the second 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, showing how a fully declarative and expressive modelling language can realise relational state–action abstractions;
2. furthermore, 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. 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 without the abstraction.

In conclusion, our results indicate that CARCASS with ASP provides a promising approach to constructing abstractions for RL, in particular when domain knowledge is available.

The rest of this paper is organised as follows. After preliminaries in Section 2, we present in Section 3 our ASP-based approach for encoding CARCASS abstractions. Sections 4 and 5 are devoted to the case studies, followed by an empirical evaluation with setup, results, and discussion in Section 6, which also touches modelling differences between ASP and Prologue. In Section 7, we review related work, and in Section 8, we conclude with directions for future work.

2 Preliminaries

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

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

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 matching the arity of f, resp., p.

A *naf-literal* is an atom $\mathbf{p}(\bar{t})$ or an expression *not* $\mathbf{p}(\bar{t})$ with negation as failure. 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 programme* $P = \{r_1, \ldots, r_n\}$ 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 programme 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 $P \vdash_{\mathrm{SLDNF}} b_1, \ldots, b_n$ (Nienhuys-Cheng and de Wolf 1997, Chapter 8).

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. In an interpretation $I \in \mathcal{I}(\Sigma)$, an atom $\mathbf{p}(\bar{t})$ is *true* if $\mathbf{p}(\bar{t}) \in I$ and *false* if $\mathbf{p}(\bar{t}) \notin I$.

We consider ASP programmes as defined in the *ASP-Core-2 input language format* (Calimeri *et al.* 2020), 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 programme P is denoted by $\mathcal{AS}(P)$.

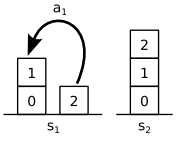
2.2 Reinforcement learning

Reinforcement Learning (Sutton and Barto 2018) 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 takes an *action* A_t. The environment transitions to a new state S_{t+1} based on the effects of the action and a *reward* R_{t+1} is obtained. The result is a history of interactions $\mathsf{H} = (\mathsf{S}_0, \mathsf{A}_0, \mathsf{R}_1, \mathsf{S}_1, \mathsf{A}_1, \mathsf{R}_2, \mathsf{S}_2, \ldots)$. 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 task environment is formalised as a *relational Markov decision process (RMDP)* (van Otterlo 2009, p. 168). Given a PL1-signature $\Sigma = (\mathit{Func}, \mathit{Pred}_S \cup \mathit{Pred}_A)$, an RMDP consists of a set $S \subseteq \mathcal{I}((\mathit{Func}, \mathit{Pred}_S))$ of states, which are Herbrand interpretations, a set $A \subseteq \mathcal{B}((\mathit{Func}, \mathit{Pred}_A))$ of actions, where each action is an atom, and a set $R \subseteq \mathbb{R}$ of rewards. For every state $s \in S$, a set A_s of admissible actions is defined. The set of admissible state-action pairs is $\Psi \doteq \{(s, a) \mid s \in S, a \in A_s\}$. The *dynamics* of the environment are defined by the probability $p(s', r \mid s, a)$ of transitioning to state s' with a reward of r after performing action a in state s, that is $\mathsf{S}_{t+1}, \mathsf{R}_{t+1} \sim p(\,\cdot \mid \mathsf{S}_t, \mathsf{A}_t)$. With the discount rate γ, an RMDP is thus a tuple $M = (S, A, R, \Psi, p, \gamma)$. We further define the logic programmes $P_s \doteq \{\ \mathbf{p}(\bar{t}). \mid \mathbf{p}(\bar{t}) \in s\ \}$ and $P_{A_s} \doteq \{\ \mathbf{p}(\bar{t}). \mid \mathbf{p}(\bar{t}) \in A_s\ \}$.

Example 1 (Blocks World).
Consider a 3*-blocks world setting (Slaney and Thiébaux 2001) as illustrated in Figure 1. 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 the agent's task, that is a state must be reached in which* B *is on top of* L*. All actions are based on* $\mathbf{move}(B, L)$*, denoting the move of block* B *to some new location* L*. The admissible actions correspond to executable moves,*

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press



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

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

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

Fig 1. A 3-blocks world RMDP. Taking action a_1 in state s_1 causes a transition to s_2.

for example $A_{s_1} = \{\mathbf{move}(2, 1), \mathbf{move}(1, 2), \mathbf{move}(1, table)\}$. *For this particular RMDP, the effects of* **move** *actions are deterministic and cause the insertion and deletion of* **on** *atoms according to the standard blocks world dynamics. The* **goal** *atoms are part of every state and are unaffected by moves. The task of the RMDP is encoded in the reward structure: the reward is* 99 *when a state is reached in which all blocks are stacked in ascending order (i.e.,* s_2*) and is* -1 *otherwise. The discount rate is* $\gamma = 1$.

A *policy* $\pi(a \mid s)$ defines the probability of taking action $a \in A_s$ in state s, that is $\mathsf{A}_t \sim \pi(\ \cdot \mid \mathsf{S}_t)$. The goal is to find an optimal policy $\pi_\star$ that maximises the expected return for all states, that is $\mathbb{E}_{p,\pi_\star}\left[\mathsf{G}_t \mid \mathsf{S}_t = s\right] \geq \mathbb{E}_{p,\pi}\left[\mathsf{G}_t \mid \mathsf{S}_t = s\right]$ for all π, s, t. A classic algorithm to find $\pi_\star$ without prior knowledge of p is *Q-learning* (Watkins and Dayan 1992). Given a step-size parameter $\alpha \in (0, 1]$ and an *action-value function* $\mathsf{Q} : \Psi \to \mathbb{R}$, the version we use (Sutton and Barto 2018, Chapter 6.5) is governed by the update

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

An approximation of $\pi_\star$ is obtained by selecting actions based on $\arg\max\{\mathsf{Q}(s, a) \mid a \in A_s\}$.

2.3 CARCASS abstractions

We now introduce the CARCASS framework (van Otterlo 2009, Chapter 5), discussing its syntax, semantics and finally an abstract version of Q-learning.

Let Σ be a PL1-signature and *Var* a set of variables.

- An *abstract state* $\hat{s}$ has the form $\hat{s} \doteq l_1, \ldots, l_k$, $k \geq 1$, with all l_i naf-literals over Σ and *Var*.
- An *abstract action* $\hat{a}$ is a single predicate atom $\hat{a} \doteq \mathbf{p}(\bar{u})$ over Σ and *Var*.
- 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 list of $n \geq 1$ elements, such that every $\hat{s}_i$ is an abstract state and every $A_{\hat{s}_i} = \{\hat{a}_{i,1}, \ldots, \hat{a}_{i,m_i}\}$ is a set of $m_i \geq 1$ abstract actions that are *admissible* in $\hat{s}_i$. The variables occurring in $A_{\hat{s}_i}$ must also occur in $\hat{s}_i$. Furthermore, we let $\hat{S} = \{\hat{s}_i \mid 1 \leq i \leq n\}$ and $\hat{\Psi} = \{(\hat{s}_i, \hat{a}_{i,j}) \mid 1 \leq i \leq n, 1 \leq j \leq m_i\}$.

The semantics of a CARCASS $\hat{C}$ is defined via a covering relation *Cov*, which uses SLDNF-resolution. Intuitively, to choose an abstract state-action pair for some concrete pair (s, a), $\hat{C}$ is interpreted as a *decision list* that is traversed until the first $\hat{s}_i$ is found which covers s. Then, some $\hat{a}_{i,j}$ with $(\hat{s}_i, \hat{a}_{i,j})$ covers (s, a) is chosen from $\hat{A}_{\hat{s}_i}$. Formally, given an RMDP $M = (S, A, R, \Psi, p, \gamma)$ and $\hat{C}$, for every $\hat{s}_i \in \hat{S}$ and $(\hat{s}_i, \hat{a}_{i,j}) \in \hat{\Psi}$:

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

Table 1. *Example CARCASS (van Otterlo 2009, p. 253)*

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

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

Example 2 (Blocks World cont'd).
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.

Algorithm 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 with a history $\mathsf{H} = (\mathsf{S}_0, \mathsf{A}_0, \mathsf{R}_1, \mathsf{S}_1, \ldots)$. 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 abstract policies and value functions.

3 ASP-based abstractions

We provide a method to encode CARCASS abstractions in ASP and show how it 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 $M = (S, A, R, \Psi, p, \gamma)$ 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.

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

Algorithm 1 Q-learning for CARCASSs (van Otterlo 2009, p. 258)

Input: RMDP $M = (S, A, R, \Psi, p, \gamma)$, 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 $\hat{\mathsf{Q}} : \hat{\Psi} \to \mathbb{R}$, step size $\alpha \in (0, 1]$
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 an abstract behaviour policy derived from $\hat{\mathsf{Q}}$
    Choose a random action $\mathsf{A}_t \in [\![\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t]\!]_{\hat{C}}^{\mathsf{S}_t}$
    Take action A_t, observe $\mathsf{S}_{t+1}, \mathsf{R}_{t+1}$
    Find $\hat{\mathsf{S}}_{t+1} \in \hat{S}$ for which $\mathsf{S}_{t+1} \in [\![\hat{\mathsf{S}}_{t+1}]\!]_{\hat{C}}$
    $\hat{\mathsf{Q}}(\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t) \leftarrow \hat{\mathsf{Q}}(\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t) + \alpha \cdot (\mathsf{R}_{t+1} + \gamma \cdot \max_{\hat{a}} \{\hat{\mathsf{Q}}(\hat{\mathsf{S}}_{t+1}, \hat{a})\} - \hat{\mathsf{Q}}(\hat{\mathsf{S}}_t, \hat{\mathsf{A}}_t))$
  until *end of episode* **or** *maximum number of steps reached*
end
return $\hat{\mathsf{Q}}$

Definition 1 (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.*

We now present our CARCASS encoding. To this end, we introduce several new atoms:

- $\mathbf{sIdx}(\lambda(\hat{s}_i), i)$ states an abstract state $\hat{s}_i$ (identified by its label) is at index i in the decision list.
- $\mathbf{sCov}(\lambda(\hat{s}_i), (\bar{T}))$ states a covering relation between $\hat{s}_i$ and some state s exists. Denoting all free variables occurring in $\hat{s}_i$ by $\bar{V} = V_1, \ldots, V_m$, $(\bar{T})$ represents a ground substitution $\bar{V}\theta = \bar{T}$, reminiscent of the original Cov, where $P_s \vdash_{\text{SLDNF}} \hat{s}_i\theta$.
- $\mathbf{sChoice}(\lambda(\hat{s}_i))$ states $\hat{s}_i$ as "chosen", such that all abstract states covering some concrete state can be enumerated as answer sets.
- $\mathbf{aCov}(\lambda(\hat{s}_i, \hat{a}_{i,j}), p(\bar{t}))$ states a covering relation between $(\hat{s}_i, \hat{a}_{i,j})$ (identified by its label) and a concrete pair (s, a) exists in each answer set where $\hat{s}_i$ is "chosen". Note that we use two representations for the covered action a: as atom $a = \mathbf{p}(\bar{t})$, as originally defined; and as the term $p(\bar{t})$ in **aCov**.

Armed with these auxiliary predicates, we develop the CARCASS encoding by combining several programmes that encode different CARCASS components.

We shall use a programme $P_{\hat{s}_i}$ that encodes a single abstract state, a programme $P_{\hat{s}_i, \hat{a}_{i,j}}$ that encodes the covering relation for abstract state-action pairs, and programmes P_{Cov} and $P_{\hat{C}}$ that translate the decision list semantics following the guess-and-check methodology. Programme P_{Cov} produces solution candidates using a choice rule, and $P_{\hat{C}}$ adds a constraint to eliminate unwanted candidates. Given a concrete state s, represented by programme P_s, and its set A_s of admissible concrete actions, represented by P_{A_s}, $P_s \cup$

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

$P_{A_s} \cup P_{Cov}$ yields one solution candidate per covering abstract state. Furthermore, $P_s \cup P_{A_s} \cup P_{\hat{C}}$ eliminates all but one such candidate, corresponding to the selected abstract state via the CARCASS semantics.

Definition 2 (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, we define programmes as follows:

- *$P_{\hat{s}_i}$ encodes the CARCASS covering relation for $\hat{s}_i$ by*

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

  *The first rule is a fact defining **sIdx** as just described. The head of the second rule defines **sCov**; its body consists of the literals in $\hat{s}_i$. The ground instances of this rule reflect all of the possible ground substitutions for the variables in $\hat{s}_i$.*
- *$P_{\hat{s}_i, \hat{a}_{i,j}}$ encodes the CARCASS covering relation for $(\hat{s}_i, \hat{a}_{i,j}) \in \hat{\Psi}$, with $\hat{a}_{i,j} = \mathbf{p}(\bar{u})$, by*

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

  *where p/n is a fresh function name of the same arity as $\mathbf{p}/n$. The head of this rule defines **aCov**; its body reflects the definition of $Cov(\hat{s}_i, \hat{a}_{i,j})$, namely the coverage of $\hat{s}_i\theta$ (by **sCov**) under the possible ground substitutions θ of the rule (in particular the ground substitutions $\bar{V}\theta$ of $\bar{V}$), and the admissibility of concrete actions $a = \mathbf{p}(\bar{u})\theta$ that are ground instances of $\hat{a}_{i,j}$ under said substitutions.*
- *P_{Cov} encodes the covering relations for all abstract states and state-action pairs by*

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

  It includes a choice rule, in which the labels of all covering abstract states are made available for the choice, but only one can be chosen per answer set.
- *$P_{\hat{C}}$ encodes the full CARCASS semantics by*

$$P_{\hat{C}} \doteq P_{Cov} \cup \{\, \leftarrow \mathbf{sChoice}(R1), \mathbf{sCov}(R2, _), \mathbf{sIdx}(R1, I1), \mathbf{sIdx}(R2, I2), I2 < I1. \,\}.$$

  The constraint added to P_{Cov} eliminates all solution candidates except for the one corresponding to the minimal covering state with respect to the decision list ordering.

Example 3 (Blocks World cont'd).
To illustrate the ASP encoding, consider the CARCASS from Example 2, in particular the abstract state $\hat{s}_2$ and the abstract pair $(\hat{s}_2, \hat{a}_{2,1})$. We choose descriptive names as labels, for example $\lambda(\hat{s}_2) = \texttt{3towers}$ and $\lambda((\hat{s}_2, \hat{a}_{2,1})) = \texttt{3t-mAB}$. Then:

$$P_{\hat{s}_2} = \left\{ \begin{array}{l} \mathbf{sIdx}(\texttt{3towers}, 2). \\ \mathbf{sCov}(\texttt{3towers}, (A, B, C)) \leftarrow \mathbf{on}(A, \mathit{table}), \mathbf{on}(B, \mathit{table}), \mathbf{on}(C, \mathit{table}), \\ \qquad\qquad\qquad\qquad\qquad\qquad A \neq B, B \neq C, A \neq C. \end{array} \right\},$$

$$P_{\hat{s}_2, \hat{a}_{2,1}} = \left\{ \begin{array}{l} \mathbf{aCov}(\texttt{3t-mAB}, \mathit{move}(A, B)) \leftarrow \mathbf{sChoice}(\texttt{3towers}), \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathbf{sCov}(\texttt{3towers}, (A, B, C)), \mathbf{move}(A, B). \end{array} \right\}.$$

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

The encodings for the other abstract states $\hat{s}_1$ and $\hat{s}_3$, and their corresponding abstract state-action pairs follow the same pattern. The encoding for P_{Cov} is the union of the ones for all abstract states and state-action pairs with the choice rule added as defined above, that is $P_{Cov} = (P_{\hat{s}_1} \cup P_{\hat{s}_1,\hat{a}_{1,1}} \cup P_{\hat{s}_1,\hat{a}_{1,2}} \cup P_{\hat{s}_1,\hat{a}_{1,3}}) \cup (P_{\hat{s}_2} \cup P_{\hat{s}_2,\hat{a}_{2,1}} \cup \ldots \cup P_{\hat{s}_2,\hat{a}_{2,6}}) \cup (P_{\hat{s}_3} \cup P_{\hat{s}_3,\hat{a}_{3,1}}) \cup \{1 = \{\mathbf{sChoice}(R) : \mathbf{sCov}(R, _)\ \}.\}$. Finally, $P_{\hat{C}}$ is defined as above. The complete encoding is available in the supplementary material.

3.2 Correctness

As for the correctness of the encoding, we establish the following result (proof in the supplementary material).

Proposition 1.
Given an RMDP $M = (S, A, R, \Psi, p, \gamma)$, a CARCASS $\hat{C}$, and labels λ, 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 $V_1, \ldots, V_m$. Furthermore, let θ be a substitution such that $\hat{s}_i\theta$ is a ground instance of $\hat{s}_i$. Then, $P_s \vdash_{\text{SLDNF}} \hat{s}_i\theta$ iff $\mathbf{sCov}(\lambda(\hat{s}_i), (V_1\theta, \ldots, V_m\theta)) \in I$.

Based on this result, desired properties of the encoding can be concluded.

Corollary 1.
For M, $\hat{C}$, λ, s, and $\hat{s}$ as in Proposition 1, the following properties (P1)–(P3) hold:

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

(P2) *Each $I \in \mathcal{AS}(P_s \cup P_{A_s} \cup P_{Cov})$, 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 $\mathbf{aCov}(\lambda(\hat{s}_i, \hat{a}_{i,j}), \mathrm{p}(\bar{t}))$ 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

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

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

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

Algorithm 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 $\hat{s}$, abstract 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**
12 **return** $\hat{s}$, $\hat{A}'_{\hat{s}}$, $\left\{[\![\hat{s}, \hat{a}]\!]^s_{\hat{C}}\right\}_{\hat{a} \in \hat{A}'_{\hat{s}}}$

3.3 Online learning for ASP-based CARCASS abstractions

Algorithm 2 illustrates how the ASP encoding of a CARCASS can be used in an online learning setting such as Algorithm 1. The ASP encoding used here includes (i) the programmes discussed so-far, (ii) an optional programme P_B with background knowledge, and (iii) the programme

$$P_{\hat{s}_\infty} \doteq \{\ \mathbf{cState}(\texttt{true}, \#sup).\quad \mathbf{cStateCovers}(\texttt{true}, ()).\ \}$$

(where $\#sup$ stands for the *supremum*, or $+\infty$) 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 $\hat{A}_{\hat{s}_\infty} \doteq \{\hat{a}_{\infty,1}\}$, such that $[\![\hat{s}_\infty, \hat{a}_{\infty,1}]\!]^s_{\hat{C}} \doteq A_s$ covers all admissible concrete actions. As for (ii), the CARCASS encoding can be combined with background knowledge where P_B may include weak constraints. However, if P_B leads to multiple (optimal) answer sets, one needs to be careful that they agree on the chosen abstract state. This can be ensured by adding the weak constraint $:\sim \mathbf{sChoice}(R), \mathbf{sIdx}(R, I).\ [I@1, R]$ for tie-breaking.

In line 1 of Algorithm 2, the augmented ASP encoding is handed to an ASP solver, which is expected to return an optimal answer set. If $P_B = \emptyset$, a single answer set corresponding to the covering abstract state with the smallest index exists, as previously described. In case of multiple optimal answer sets, the solver chooses one to return. 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 if 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 $\hat{a} \in \hat{A}'_{\hat{s}}$.

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

4 Case study: Blocks world

As a first application, we discuss n-blocks world stacking tasks and present the encoding of a state-action pair abstraction, inspired by the algorithms GN1 and GN2 (Slaney and Thiébaux 2001), where the size of the abstract state-action space is constant in n. 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 the supplementary material.

The abstraction was designed with a generalised version of the RMDP from Example 1 in mind. We assume an n-blocks world RMDP with an arbitrary number of $n \geq 1$ blocks and an arbitrary stacking task, encoded in the reward structure and communicated to the agent via the set of **goal** atoms in the state descriptions.

Several predicates were defined to support the abstraction, including the following: $\mathbf{clear}(L)$ denotes that location L is clear, that is 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 on top of B to fulfil the stacking task. Note that the definition of **incomplete** depends on both **on** and **goal**.

Example 4 (Example 1 cont'd).
We augment s_1 *with the facts* $\mathbf{clear}(\mathit{table})$, $\mathbf{clear}(1)$, $\mathbf{clear}(2)$, $\mathbf{above}(0, \mathit{table})$, $\mathbf{above}(1, 0)$, $\mathbf{above}(1, \mathit{table})$, $\mathbf{above}(2, \mathit{table})$, *and* $\mathbf{incomplete}(\mathit{table}, 1)$. *To further illustrate incomplete towers, consider a different* 5*-blocks world RMDP with a (partial) stacking task where blocks* 0–3 *need to form a tower in ascending order and the position of block* 4 *is irrelevant. The corresponding state*

$$s_3 = \{\mathbf{on}(4, \mathit{table}), \mathbf{on}(0, 4), \mathbf{on}(1, 0), \mathbf{on}(2, 1), \mathbf{on}(3, \mathit{table}), \mathbf{goal}(1, 0), \mathbf{goal}(2, 1), \mathbf{goal}(3, 2)\}$$

is augmented with $\mathbf{incomplete}(0, 2)$, *reflecting the fact that blocks* 0, 1, 2, *and* 4 *are in their final position, but* 3 *must still be added on top of* 2. *For state* $s_3 \cup \{\mathbf{goal}(4, 3)\}$ *in yet another* 5*-blocks world RMDP with the task to stack all five blocks in ascending order, no goal tower exists since no blocks are in their final position.*

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, that is 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

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

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

The next move towards fulfilling the stacking task 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

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

We also define $\hat{a}_{i,\infty}$ as a catch-all abstract action, covering actions not covered by $\hat{a}_{0,0}$, encoded as

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

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

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

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

$$P_{\hat{s}_1,\hat{a}_{1,0}} = \{\, \mathbf{aCov}(move(b,table), move(B,table)) \leftarrow \mathbf{sChoice}(r1), \mathbf{sCov}(r1,(_,_,B)).\,\}$$

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

To advance $\hat{s}_1$ towards fulfilling the stacking task, all blocks on top of T must 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 $\hat{a}_{i,\infty}$.

The intuition behind this abstraction relates back to the algorithms GN1 and GN2 (Slaney and Thiébaux 2001). 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.

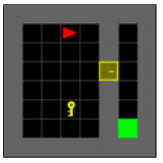
5 Case study: MiniGrid

Next, we consider environments from the MiniGrid simulation library (Chevalier-Boisvert *et al.* 2023). 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 (Figure 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, 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 must 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

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press



$$s_1 = \left\{ \begin{array}{l} \textbf{obj}(goal, (6,6)) \\ \textbf{obj}(door(yellow, locked), (5,3)) \\ \textbf{obj}(key(yellow), (3,5)) \\ \textbf{obj}(agent(east), (3,1)) \\ \textbf{obj}(wall(grey), (0,0)) \\ \textbf{obj}(wall(grey), (0,1)) \\ \ldots \end{array} \right\}$$

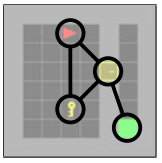


Fig 2. Example Minigrid state: The left image shows a state, its relational representation is in the centre, and a graph of the room layout, inferred using ASP, is to the right.

contains only information relevant to the first node on the path. The full encoding of the abstraction is available in the supplementary material.

We assume a MiniGrid configuration in which the current state is fully observable, resembling a birds-eye view as in Figure 2, and replace the default state description (a matrix $\mathbf{A} \in \mathbb{N}_0{}^{w \cdot h \cdot 3}$, where $w \times h$ is the size of the grid world) with a relational state description, using the following atoms: $\textbf{obj}(O, (X, Y))$ asserts that object O is located at the grid coordinates (X, Y); $\textbf{carries}(O)$ 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**.

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

Example 6.
The computed path in s_1 *(Figure 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* $\textbf{oriented}(east)$, $\textbf{fronting}((4,1))$, $\textbf{xdir}(onAxis)$, $\textbf{ydir}(south)$, $\textbf{touching}(none)$, *and* $\textbf{ingap}(none)$.

We use 757 abstract states (excluding $\hat{s}_\infty$): $\hat{s}_1$ covers potentially dangerous situations, encoded as

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

while the states $\hat{s}_2$–$\hat{s}_{757}$, representing all possible combinations of ground instances for **oriented**, **xdir**, **ydir**, **touching**, and **ingap**, are compactly encoded as

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

The background knowledge is designed such that every answer set can have at most one instance of **label**, 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.

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

While actions can simply be stated as facts, for example **aCov**(*left*, *left*)., we can prune actions that are admissible in the concrete representation but have no effect (i.e., do not lead to state changes). This can be done by omitting such actions (i.e., making them inadmissible) in the abstract representation. For example, **aCov**(*pickup*, *pickup*) ← **fronting**(T), **obj**$(key(_), T)$ ensures that the **pickup** action is admissible in the abstract representation 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 space and sample needs of the learning process. Notably, there is no guarantee of convergence of abstract Q-learning in general nor that a learned abstract policy is optimal. Our experiments are thus designed to shed light on **(E1)** the *stability of the learning process*, **(E2)** the *quality of the learned policy*, and **(E3)** the *differences in sample efficiency* between concrete and abstract Q-learning.

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 full details, including platform, software, and parameter settings see the supplementary material.

As for **(E1)**, we observe the return of episodes across the learning process. Concerning **(E2)**, we measure for every realised episode h_i the solution quality by the *return* $\mathsf{G}_0(h_i)$ at time 0. We call an episode *successful* if it achieved a return above a baseline, viz. $\mathsf{G}_0(h_i) > 0$ for MiniGrid and $\mathsf{G}_0(h_i) \geq 62$ for Blocks World (i.e., the worst-case return of a naive unstack-stack strategy). For measuring **(E3)** sample efficiency, we use 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. Figure 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 after episode 2000 for merely 10 instances Q_1 dropped below the baseline of 62, and for none below 60. For Door Key, Multi Room, and Four Rooms, 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 and the percentage of successful episodes over all realised, resp., 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.

(E1) 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, the majority of episodes realised by abstract Q-learning are unaffected. Still, stability issues may be present in less than 25% of experiment repetitions.

(E2) Regarding *quality*, the high success rates of episodes over the last 500 episodes suggest that episodes of acceptable quality can be learned with high consistency. We

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

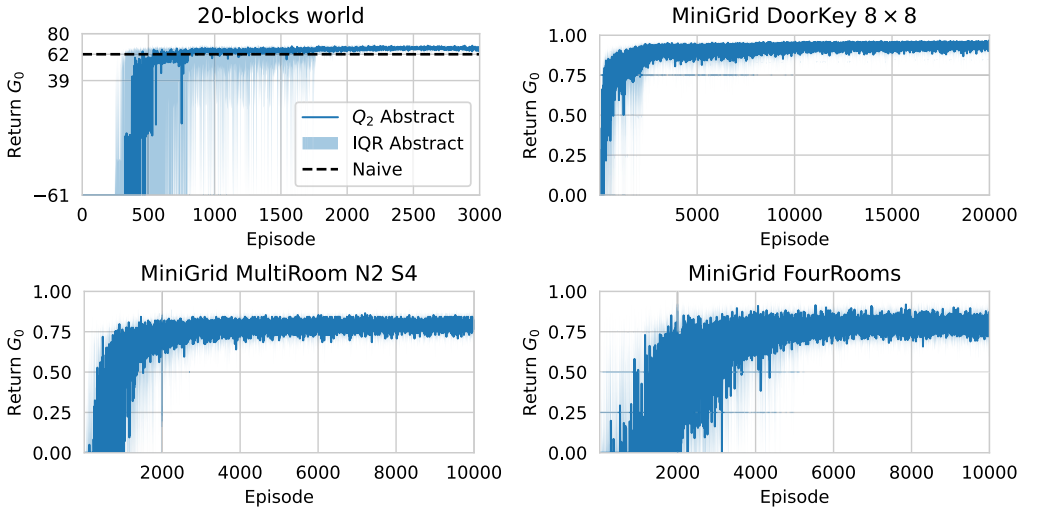


Fig 3. The learning curves of abstract Q-learning in four task environments.

cannot make statements about the optimality of the learned policies. However, for the 20-blocks world, there are still episodes where the 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.

(E3) For *sample efficiency*, abstract Q-learning improves on sample efficiency over concrete Q-learning bases on the differences in the mean returns and the percentages of successful episodes over the entire learning processes.

In conclusion, abstract Q-learning reliably produced policies of acceptable quality and, moreover, *in a smaller number of episodes* as compared to concrete Q-learning.

6.1 Comparison with prologue-based abstractions

We also developed Prologue encodings (to be found in the supplementary material) to contrast the modelling styles encouraged by the two formalisms for the considered relational decision-making problems.

The Prologue encoding of the Blocks World CARCASS was relatively simple to obtain, as it is almost identical to the ASP encoding. The differences in the modelling approaches become more apparent in the Minigrid abstraction, which can be viewed as a shortest-path problem under relational constraints. In the ASP encoding, admissible paths and optimality criteria are specified declaratively using choice rules, constraints, and optimisation statements, resulting in a compact encoding that closely follows the problem description. In the Prologue encoding, solving the same abstraction requires an explicit implementation of the underlying search process (in our case, a breadth-first search), leading to a more algorithmic representation in which aspects of control and problem specification are more explicitly represented. Further details are provided in the supplementary material.

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

Table 2. *Comparison of abstract and concrete Q-learning (QL) in four task environments*

Environment	Repr.	Mean $\mathsf{G_0}$ (all eps.)			# Succ. (all eps.)			# Succ. (last 500)		
		$\mathbf{Q_1}$	$\mathbf{Q_2}$	$\mathbf{Q_3}$	$\mathbf{Q_1}$	$\mathbf{Q_2}$	$\mathbf{Q_3}$	$\mathbf{Q_1}$	$\mathbf{Q_2}$	$\mathbf{Q_3}$
20-blocks world	Abstract QL	18.64	43.86	54.07	0.40	0.76	0.85	0.93	0.97	0.98
	Concrete QL	−61.00	−61.00	−61.00	0.00	0.00	0.00	0.00	0.00	0.00
Door Key	Abstract QL	0.82	0.86	0.90	0.93	0.97	0.99	1.00	1.00	1.00
	Concrete QL	0.01	0.01	0.01	0.02	0.02	0.03	0.02	0.02	0.04
Four Rooms	Abstract QL	0.59	0.61	0.64	0.78	0.81	0.84	1.00	1.00	1.00
	Concrete QL	0.02	0.02	0.02	0.03	0.03	0.04	0.03	0.04	0.04
Multi Room	Abstract QL	0.68	0.69	0.70	0.91	0.91	0.92	1.00	1.00	1.00
	Concrete QL	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03

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 programmes* (LOMDPs) (Kersting and De Raedt 2004), which describe abstract states as logical rules and transitions as probabilistic effects of actions, and *relational Q-learning* (Morales 2003), which groups states and actions into logical templates that define when an abstract action can be applied. Other extensions include *probabilistic relational models* (PRMs) (Guestrin 2003), which decompose value functions across object classes, work on *stochastic policies for relational POMDPs* (Itoh 2004), where policies are expressed as decision lists with probabilities, and a learning classifier system in first-order logic (FOXCS) (Mellor 2007). All of these approaches share with *CARCASS* (van Otterlo 2009) 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 Prologue using SLDNF-resolution. Like CARCASS, FOXCS represents policies via sets of rules, implemented in Prologue, that can generalise over states and actions. Unlike CARCASS, the head of each rule consists of exactly one abstract action (not multiple) and the set of rules is not interpreted as a decision list, but individual rules advocate for their covered actions in a weighted voting process. Our work continues the CARCASS line of research but replaces Prologue 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* but quite differently. For instance, relational meta-policies have been encoded in ASP for hierarchical option selection (Mitchener *et al*. 2022), and other works use ASP to restrict exploration by pruning infeasible actions, for example Ferreira *et al*. (2017). ASP was also used to encode entire relational MDPs, including states and value functions (Saad 2011). However, none of these

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

works proposed a CARCASS-style general ASP-based representation of *full state-action abstractions*, which is our core contribution.

8 Conclusion

We investigated ASP for modelling 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 Prologue-based CARCASS abstractions.

Future work includes automatic generation and refinement of ASP abstractions (Saribatur *et al*. 2021), leveraging non-determinism and optimisation to guide exploration, and testing transfer of abstract policies to larger or previously unseen tasks. 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.

Supplementary material

To view supplementary material for this article, please visit https://doi.org/10.1017/S1471068426100544.

Competing interests

The authors declare none.

References

Brewka, G., Eiter, T. and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103. https://doi.org/10.1145/2043174.2043195.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. 2020. ASP-Core-2 input language format. *Theory and Practice of Logic Programming* 20, 2, 294–309. https://doi.org/10.1017/S1471068419000450.

Chevalier-Boisvert, M., Dai, B., Towers, M., de Lazcano, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S. and Terry, J. 2023. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. CoRR, abs/2306.13831. https://doi.org/10.48550/ARXIV.2306.13831.

Ferreira, L. A., Bianchi, R. A. C., Santos, P. E. and de Mántaras, R. L. 2017. Answer set programming for non-stationary markov decision processes. *Applied Intelligence* 47, 4, 993–1007. https://doi.org/10.1007/S10489-017-0988-Y.

Guestrin, C. E. 2003. *Planning under Uncertainty in Complex Structured Environments*. Stanford University.

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press

Itoh, H. 2004. 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)*, 34–39.

Kersting, K. and De Raedt, L. 2004. Logical Markov decision programs and the convergence of logical TD(lambda). In *Inductive Logic Programming, 14th International Conference, ILP 2004, Proceedings*. Porto, Portugal, *September 6-8, 2004*, R. Camacho, R. D. King and A. Srinivasan, Eds. Lecture Notes in Computer Science, vol. 3194. Springer, 180–197. https://doi.org/10.1007/978-3-540-30109-7_16.

Mellor, D. 2007. A learning classifier system approach to relational reinforcement learning. In *Learning Classifier Systems, 10th International Workshop, IWLCS 2006, Seattle, MA, USA, July 8, 2006 and 11th International Workshop, IWLCS 2007, London, UK, July 8, 2007, Revised Selected Papers*. J. Bacardit, E. Bernadó-Mansilla, M. V. Butz, T. Kovacs, X. Llorà and K. Takadama, Eds. Lecture Notes in Computer Science. Springer, 169–188.

Mitchener, L., Tuckey, D., Crosby, M. and Russo, A. 2022. Detect, understand, act: A neuro-symbolic hierarchical reinforcement learning framework. *Machine Learning* 111, 4, 1523–1549. https://doi.org/10.1007/S10994-022-06142-7.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540, 529–533.

Morales, E. F. 2003. Scaling up reinforcement learning with a relational representation. In *Proceedings of the Workshop on Adaptability in Multi-Agent Systems at AORC 2003, Sydney, Australia*.

Nienhuys-Cheng, S. and de Wolf, R. 1997. *Foundations of Inductive Logic Programming*. Lecture Notes in Computer Science, vol. 1228. Springer. https://doi.org/10.1007/3-540-62927-0. ISBN: 3-540-62927-0.

Saad, E. 2011. Bridging the gap between reinforcement learning and knowledge representation: A logical off- and on-policy framework. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 11th European Conference, ECSQARU 2011. Proceedings*, Belfast, UK, *June 29–July 1, 2011*, W. Liu, Ed. LNCS, vol. 6717. Springer, 472–484. https://doi.org/10.1007/978-3-642-22152-1_40.

Saribatur, Z. G., Eiter, T. and Schüller, P. 2021. Abstraction for non-ground answer set programs. *Artificial Intelligence* 300, 103563. https://doi.org/10.1016/J.ARTINT.2021.103563.

Slaney, J. K. and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125, 1–2, 119–153. https://doi.org/10.1016/S0004-3702(00)00079-5.

Sutton, R. S. and Barto, A. G. 2018. *Reinforcement Learning - An Introduction*. 2nd ed. MIT Press.

van Otterlo, M. 2004. Reinforcement learning for relational MDPs. In *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands (BeNeLearn 2004)*, *January 8–9, 2004*, A. Nowé, T. Lenaerts and K. Steenhaut, Eds., 138–145.

van Otterlo, M.2005. A survey of reinforcement learning in relational domains. Technical Report 05-31, Centre for Telematics and Information Technology (CTIT), Univ. Twente, Enschede, The Netherlands. [Accessed on August 20, 2025].

van Otterlo, M. 2009. *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty in First-Order and Relational Domains*, vol. 192. IOS Press.

Watkins, C. J. C. H. and Dayan, P. 1992. Q-learning. *Machine Learning* 8, 3, 279–292. https://doi.org/10.1007/BF00992698. ISSN: 1573-0565.

https://doi.org/10.1017/S1471068426100544 Published online by Cambridge University Press