EXTRACTING SYMBOLIC GENERAL POLICIES FROM BOOLEAN RELATIONAL GNNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Relational graph neural networks (R-GNNs) learn policies for classical planning that generalize to much larger instances than those seen in training, but the policies are opaque, and it is unclear why they generalize so well. We shed light on this question by extracting symbolic policies from R-GNNs with max aggregation and binary object embeddings. We show that every layer of such a network can be described in description logic: the update of each object's embedding is a Boolean function of a finite set of $\mathcal{ALCI}$ features derived from the network itself. Decoding a layer thus reduces to finding, for each embedding vector, a minimal DNF over these features, which we solve exactly with SAT and MaxSAT. The resulting concepts are consistent with the network on all sampled training states by construction, and provably minimal whenever the search finishes within its budget. Rules over changes in concept counts then replace the readout, separating the transitions the network considers good from the others. We further prove that instances up to some fixed size already contain every computation the network performs on an object of any instance, however large, with explicit bounds for Gripper and Ferry. Across 12 domains, the symbolic policies solve 94.6% of the test instances, on par with the 93.6% of the R-GNN policies, and remain optimal wherever the R-GNN policy is. Our concepts are $4.4\times$ smaller than those of a greedy decision-tree decoder, and in some cases the policies are readable and provably optimal on every instance. The results illustrate that R-GNNs generalize across planning instances because, regardless of the number of objects, they map objects into finitely many classes characterized by logical concepts that support general policies.

1 INTRODUCTION

Generalized planning aims to find policies that solve all instances of a planning domain, whatever their initial states, goals, and number of objects (Srivastava et al., 2011; Hu & Giacomo, 2011; Jiménez et al., 2019). There are two main approaches. The first consists of symbolic methods, which generate a pool of candidate features, typically description-logic concepts over the domain's predicates, and use combinatorial search to select the features that define a policy (Khardon, 1999; Boutilier et al., 2001; Martín & Geffner, 2004; Fern et al., 2006; Sanner & Boutilier, 2009; Bonet & Geffner, 2018; Francès et al., 2021; Drexler et al., 2022). The resulting policies are easy to read and often come with correctness guarantees, but the feature search is expensive. The second consists of learning methods, mainly graph neural networks (GNNs) over relational state representations, which scale much further and learn strong policies from data (Toyer et al., 2020; Bajpai et al., 2018; Rivlin et al., 2020; Ståhlberg et al., 2022a; 2023; Chen et al., 2024). However, the learned policies are black boxes: it is hard to understand their rules, verify their correctness, or explain why they generalize to much larger instances (Chakraborti et al., 2021).

In this paper, we address these questions *after* training, by extracting a symbolic policy from the trained network. We use relational GNNs (R-GNNs) with binary object embeddings, so that each layer assigns every object one of finitely many embedding vectors, and ask: *which description-logic concept describes the objects that receive a given embedding vector?* Existing methods build such concepts from *features*, simpler concepts such as "is clear" or "is on a clear block" taken from a *feature pool*, and face two problems: choosing the pool, and combining its features into a small concept. Symbolic policy learning generates a large pool from a grammar and searches it,

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

learning only afterwards whether the pool contains the right features (Francès et al., 2021). SAT-based concept learners search all concepts up to a size bound, at a cost exponential in the size (ten Cate et al., 2023; Funk et al., 2026). Greedy logical distillation extracts a GNN layer by layer into decision trees over a feature pool, with no guarantee that the result is small (Pluska et al., 2024).

We resolve both problems for these networks. First, the feature pool need not be designed or searched for: it can be read off the network. A layer computes an object's new embedding only from its own previous embedding and the messages sent by the atoms it occurs in, and since messages are combined by a maximum, as in the planning R-GNNs we build on (Ståhlberg et al., 2022a), only *which* messages arrive matters, not how many. The new embedding is thus determined by a few yes/no questions about the object: which embedding did it have in the previous layer, which unary predicates hold for it, and, for each binary predicate and previous embedding, does it have a neighbor with that embedding? Once the previous layer has been decoded into concepts C_x, each question is itself an $\mathcal{ALCI}$ concept, such as $\exists \mathsf{on}.C_x$, so we obtain a finite pool that provably determines the layer's embeddings (Theorem 1). Second, since this pool is small, we can find a minimal combination of its features exactly.

In practice, we sample states from small instances and, for each embedding vector, use SAT and MaxSAT to find a minimal DNF (disjunctive normal form) formula over the features that is true exactly on the objects with that vector; planning states are so constrained that only a few distinct feature vectors occur. The resulting concepts are consistent with the network on all sampled states, and provably minimal whenever the search finishes within its budget. Rules over changes in concept counts then replace the network's readout. The resulting policies are readable: on `gripper`, for example, the first rule tests whether the number of "undelivered goal balls" or "rooms holding goal balls" decreases, and we prove that the policy is optimal on every Gripper instance (Section 6.3).

We show that, for a fixed domain and number of layers, a finite *complete* training set contains every computation the encoder performs on an object of any instance, for every choice of weights, with explicit sizes for Gripper and Ferry (Section 5). This explains why concepts decoded from small instances work on larger ones, and indeed we never see a new embedding vector during testing (Appendix C.4).

Our contributions are:

- A **Boolean relational GNN** architecture (Section 3) that can be trained in the standard way but whose embeddings can be understood logically.
- A **logical interpretation** of the Boolean R-GNN object embeddings (Section 4) in the description logic $\mathcal{ALCI}$ that is provably consistent with the network on all sampled states and guaranteed to be minimal whenever the search finishes within its budget; this is a concrete, per-network version of known expressiveness results (Barceló et al., 2020).
- A **proof** that bounded-size training instances cover every computation of the network on an object of any instance (Section 5), with explicit bounds for Gripper and Ferry.
- Experiments showing that the **symbolic policies match the network** (Section 6): on 12 domains, they solve 94.6% of the test instances against 93.6% for the network, with concepts $4.4\times$ smaller than those of a greedy decoder; the Gripper policy is provably optimal, and L1 regularization yields simpler concepts.

2 Background

Classical and generalized planning. A classical planning instance is a tuple $P = \langle D, \mathcal{O}_P, s_0, g_P \rangle$, where the *domain* $D = \langle \mathcal{P}, \mathcal{A} \rangle$ fixes predicates $\mathcal{P}$ and action schemas $\mathcal{A}$, and the objects $\mathcal{O}_P$, initial state s_0, and goal g_P are specific to the instance; we drop the subscript P when the instance is clear. A *state* s is a set of ground atoms over $\mathcal{P}$ and $\mathcal{O}$, and satisfies the goal if $g \subseteq s$. An action a is applicable in s if its preconditions hold in s, and applying it yields the successor s' by removing its delete effects and adding its add effects. A *plan* is a sequence of applicable actions from s_0 to a goal state. A state is *reachable* in P if it results from s_0 by zero or more actions; $\mathrm{Reach}(P)$ is the set of reachable states. A *generalized planning* problem is a class $\mathcal{Q}$ of instances that share the domain D but differ in objects, initial states, and goals. A *training example* is a pair (P, s) with $P \in \mathcal{Q}$ and $s \in \mathrm{Reach}(P)$, and a *training set* is a finite set of training examples. The

objective is a *general policy* π that does not depend on object names, selects a successor in every non-goal state, and, followed from s_0, reaches a goal state in every instance of $\mathcal{Q}$ (Srivastava et al., 2011; Bonet & Geffner, 2018). Such policies are typically learned from small training instances and evaluated on larger ones. We consider instances whose goal is a single atom, as in our benchmarks.

Description logics. Description logics (DLs) are logic-based knowledge-representation languages that balance expressivity and decidability (Baader et al., 2003). They are built from *concepts* (unary predicates) and *roles* (binary predicates) and, like planning formalisms, separate general knowledge (the TBox) from instance data (the ABox), which makes them easy to adapt to planning. We use the fragment $\mathcal{ALCI}$, whose concepts are formed from atomic concepts and $\top, \bot$ by negation, conjunction, disjunction, and the restrictions $\exists r.C$ and $\forall r.C$, where a role r may also be the inverse p^- of a binary predicate p, relating the same pairs in the opposite direction. A state s with goal g over objects $\mathcal{O}$ is read as an interpretation: each unary predicate $p \in \mathcal{P}$ is an atomic concept with extension $[\![p]\!]_s = \{o : p(o) \in s\}$, and each binary predicate is a role. For the goal, previous work adds a goal version p_g of each predicate, read off g rather than s; we instead use the network's goal predicates (Section 3): $p_{g,t}$ for goal atoms true in s and $p_{g,f}$ for those that are false. Predicates of arity ≥ 3 are neither concepts nor roles; none of our benchmark domains uses one. We write $[\![C]\!]_s \subseteq \mathcal{O}$ for the extension of concept C in state s, leaving the fixed goal implicit, and $|C|$ for the *size* (complexity) of C, counting abstract-syntax-tree nodes: atoms and $\top/\bot$ cost 1; $\neg$, $\exists r.$ and $\forall r.$ cost 1 plus their child; $\sqcap$ and $\sqcup$ cost 1 plus the sum of their children.

3 Relational GNNs with binary object embeddings

We use the relational GNN (R-GNN) of Ståhlberg et al. (2022a), a GNN for relational structures, trained as a transition classifier that serves as the policy. We view it as an *encoder*, whose L message-passing layers map each object to an *embedding* that summarizes its relational context, followed by a *readout*, which computes the network's output from the embeddings of a state. We assume a trained R-GNN with binary embeddings and aim to make sense of its embeddings and its policy.

Input. For a state s and a goal g, the encoder reads the atoms $\mathrm{Input}(s, g)$: all atoms of s and, for each goal atom $p(o_1, \ldots, o_{\kappa_p}) \in g$, the atom $p_{g,t}(o_1, \ldots, o_{\kappa_p})$ if it is true in s and $p_{g,f}(o_1, \ldots, o_{\kappa_p})$ otherwise. We write $\mathcal{P}^*$ for the finite set of predicates that can occur in $\mathrm{Input}(s, g)$, and κ_p for the arity of $p \in \mathcal{P}^*$.

Encoder. Every object starts with the embedding $\mathbf{x}^{(0)}(o) = \mathbf{0} \in \{0, 1\}^k$. Layer $t \in \{1, \ldots, L\}$ computes $\mathbf{x}^{(t)}$ from $\mathbf{x}^{(t-1)}$ in three steps.

1. Messages. Each atom $a = p(o_1, \ldots, o_{\kappa_p}) \in \mathrm{Input}(s, g)$ sends one message to each argument; $m^{(t)}_{a,i}$ denotes the message to its i-th argument o_i. Each predicate p has its own message MLP, which outputs one fixed-size message vector per argument:

$$\big(m^{(t)}_{a,1}, \ldots, m^{(t)}_{a,\kappa_p}\big) = \mathrm{MLP}_p\big(\mathbf{x}^{(t-1)}(o_1), \ldots, \mathbf{x}^{(t-1)}(o_{\kappa_p})\big). \tag{1}$$

2. Aggregation. Each object collects the *set* of messages it receives, so repeated messages count once, and takes their componentwise maximum:

$$S^{(t)}(o) = \big\{m^{(t)}_{a,i} : a \in \mathrm{Input}(s, g),\ o_i = o\big\}, \qquad m^{(t)}(o) = \max S^{(t)}(o). \tag{2}$$

The set $S^{(t)}(o)$ is never empty: every object has the PDDL type object and thus receives at least the message of the atom $\mathsf{object}(o) \in \mathrm{Input}(s, g)$.

3. Update. An update MLP combines the old embedding with the aggregated message, and a binarizer bin maps the result to $\{0, 1\}^k$:

$$\mathbf{x}^{(t)}(o) = \mathrm{bin}\Big(\mathrm{MLP}_U\big(\mathbf{x}^{(t-1)}(o),\ m^{(t)}(o)\big)\Big). \tag{3}$$

The binarizer is a deterministic straight-through quantizer (Bengio et al., 2013): it thresholds each dimension to 0 or 1, in training and at evaluation alike, and passes gradients through unchanged. The inputs of each MLP are concatenated, and the MLPs are shared across layers.

Readout, training, and the neural policy. The network is a *transition classifier*. Given a state s, a candidate successor s' and the goal g, the encoder embeds both states and the readout sums the final embeddings of each state's objects,

$$\mathbf{E}(s) = \sum_{o \in \mathcal{O}} \mathbf{x}^{(L)}(o) \in \mathbb{N}^k.$$

A readout MLP then gives the logit $\mathrm{MLP}_R\big(\mathbf{E}(s), \mathbf{E}(s')\big)$ that $s \to s'$ is *good*, i.e. lies on an optimal path to the goal: $V^*(s') = V^*(s) - 1$, where $V^*(s)$ is the optimal plan length from s. We train on transitions from the full state spaces of the training instances with the loss $\mathcal{L}_{\mathrm{BCE}} + \lambda\|\theta\|_1$, where $\mathcal{L}_{\mathrm{BCE}}$ is the binary cross-entropy and θ are all weights of MLP_p, MLP_U, and MLP_R except the biases (Appendix C.6 compares values of λ). The induced *neural policy* greedily moves to the successor with the highest score.

4 Extracting symbolic policies

We are given a trained R-GNN transition classifier (Section 3) and want to replace both its parts, the encoder and the readout, by symbolic counterparts. All concepts are computed from states sampled from small instances; Section 5 explains why this can suffice.

For the encoder, let $X^{(t)} \subseteq \{0,1\}^k$ be the set of embedding vectors it produces at layer t on the sampled states, with $X^{(0)} = \{\mathbf{0}\}$. For every layer $t \in \{1, \ldots, L\}$ and vector $x \in X^{(t)}$, we look for a concept C_x that describes exactly the objects receiving x.

Definition 1 (Decoding problem)**.** Given a layer t and a vector $x \in X^{(t)}$, find a concept C_x of minimal size such that, for every sampled state s and every object o of s,

$$o \in [\![C_x]\!]_s \iff \mathbf{x}^{(t)}(o) = x.$$

If this holds, we say that C_x is *consistent* with the network.

We solve this problem layer by layer in two steps, and then turn to the readout. Section 4.1 identifies the features the concepts of layer t need and shows that a finite set of features built from the concepts of layer $t-1$ always suffices. Section 4.2 uses these features to find minimal concepts exactly with SAT and MaxSAT, and states what the resulting concepts guarantee. Section 4.3 replaces the readout by rules over the changes in how many objects satisfy each last-layer concept. Appendix B.1 works through the decoding of one embedding vector.

Our decoding problem is similar to that of Pluska et al. (2024), who also extract logical formulas from GNNs. A key difference is that their formulas explain the structure of the graph, e.g. how many nodes have a given label, and remain fairly opaque, while ours are readable logical interpretations of object embeddings; the Boolean embeddings of our R-GNNs make this possible. We also guarantee that the extracted concepts are minimal whenever the search finishes within its budget, and show that the decoded concepts can generalize to larger instances.

4.1 Features that suffice

A key insight of our approach is that decoding layer t builds on the concepts of layer $t-1$. The network computes an object's embedding at layer t only from embeddings at layer $t-1$: its own and its neighbors' (Equations 1–3). Once we have concepts $C_{x'}$ for the embeddings of layer $t-1$, with $C_{\mathbf{0}} = \top$ at layer 1, we can describe layer t in terms of these concepts and one more step to the neighbors. Each decoding problem then needs only concepts of quantifier depth one over the concepts of layer $t-1$, whereas decoding the last layer directly over the domain's predicates would need much deeper and larger concepts. The result is a layered set of concept definitions, a TBox in DL terms, which can be unfolded into plain $\mathcal{ALCI}$ concepts when needed.

Concretely, let $\mathcal{U}$ be the set of unary predicates and $\mathcal{R}$ the set of binary predicates and their inverses p^-, both including the goal predicates $p_{g,t}$ and $p_{g,f}$ of Section 3. The *features* of layer t are

$$\mathcal{L}^{(t)} = \underbrace{\big\{C_x : x \in X^{(t-1)}\big\}}_{\text{own previous embedding}} \cup \underbrace{\big\{p : p \in \mathcal{U}\big\}}_{\text{unary predicates}} \cup \underbrace{\big\{\exists r.C_x : r \in \mathcal{R},\ x \in X^{(t-1)}\big\}}_{\text{neighbors}}. \tag{4}$$

They answer three kinds of yes/no questions about an object: which embedding it had in the previous layer, which unary predicates hold for it, and, for each role and previous embedding, whether it has such a neighbor. The truth values of all features of layer t at an object o form its *feature vector*, where C_x is read as the set of objects with embedding x at layer $t-1$.

The following theorem shows that these features suffice: they contain everything the network uses to compute the embeddings of layer t.

Theorem 1 (Sufficiency of the feature pool)**.** *For domains whose predicates have arity at most two, the features $\mathcal{L}^{(t)}$ determine the embeddings of layer t. That is, let o and o' be objects of states whose layer-$(t-1)$ embeddings lie in $X^{(t-1)}$; if every feature in $\mathcal{L}^{(t)}$ has the same truth value at o and at o', then $\mathbf{x}^{(t)}(o) = \mathbf{x}^{(t)}(o')$.*

The proof (Appendix A.1.1) follows the intuition of the introduction: the update of Equation 3 sees only the object's previous embedding and its set of messages, which, due to the maximum in Equation 2, depends only on which kinds of neighbors the object has. The condition holds on the sampled states, which is all the decoder needs; on a larger instance it fails if a neighbor has an embedding outside $X^{(t-1)}$, which Section 5 discusses and which never happens in our experiments (Appendix C.4). Two consequences follow. First, the decoding problem always has a solution: whether an object has embedding x is a Boolean function of its features, and every Boolean function has a DNF over them. Second, only feature vectors matter, so decoding is a problem over Boolean vectors.

4.2 Finding minimal concepts

For $x \in X^{(t)}$, the *positive examples* are the feature vectors of the sampled objects o with $\mathbf{x}^{(t)}(o) = x$, and the *negative examples* those of all other sampled objects. By Theorem 1, no feature vector is both. A DNF over $\mathcal{L}^{(t)}$ is consistent with the network exactly when it is true on every positive and false on every negative example. We find a minimal such DNF in two steps, following the classical scheme for minimizing Boolean formulas (Brayton et al., 1984). We first list the *prime implicants*: the conjunctions of features and negated features that are false on every negative example but no longer are if any literal is dropped. We then choose a cheapest set of them that together are true on every positive example, where a conjunction c costs its size $|c|$; this is weighted set cover, which we solve with MaxSAT. The result C_x is the disjunction of the chosen prime implicants.

Proposition 1 (Consistency and minimality)**.** *Suppose that the concepts of layer $t-1$ are consistent with the network. Then every concept C_x of layer t is consistent with the network. If, in addition, all prime implicants are found, then the total size of the conjunctions of C_x is the smallest among all DNFs over $\mathcal{L}^{(t)}$ that are consistent with the network.*

By induction over the layers, all decoded concepts are thus consistent with the network on the sampled states; the proof is in Appendix A.1.2. Since finding all prime implicants can take exponential time, we stop the enumeration after a fixed number of them; the resulting concept is still consistent with the network, but not certified minimal (Appendix C.3). Appendix B describes the implementation, including the SAT and MaxSAT encodings.

4.3 Rules for the readout

The readout decides whether a transition $s \to s'$ is good from $\mathbf{E}(s)$ and $\mathbf{E}(s')$. Let $\mathcal{C} = (C_1, \ldots, C_m)$ be the decoded concepts of the last layer L, for the embedding vectors $x_1, \ldots, x_m$. The *count vector* $\mathbf{n}_{\mathcal{C}}(s) \in \mathbb{N}^m$ of a state s contains, for each concept, the number of objects of s satisfying it: $\mathbf{n}_{\mathcal{C}}(s)[i] = |[\![C_i]\!]_s|$. Since the concepts are consistent with the network, $\mathbf{E}(s) = \sum_i \mathbf{n}_{\mathcal{C}}(s)[i]\, x_i$, so the readout sees a state only through these counts. We describe a transition by the change in counts, $\Delta(s, s') = \mathbf{n}_{\mathcal{C}}(s') - \mathbf{n}_{\mathcal{C}}(s)$, and in particular by the signs of these changes, its *effect pattern*:

$$\sigma(s, s') = \operatorname{sgn}\big(\Delta(s, s')\big) \in \{-1, 0, 1\}^m,$$

where the sign is taken entrywise. We write Δ_i for $\Delta(s, s')[i]$, the change in the count of C_i, so the rules below test $\Delta_i > 0$, $\Delta_i < 0$ and $\Delta_i = 0$. We then learn a classifier $g\colon \{-1, 0, 1\}^m \to \{\text{good}, \text{bad}\}$ from sampled transitions labeled good or bad by the network, and the symbolic policy

moves from s to a successor s' with $g(\sigma(s, s')) = \text{good}$. We use a decision tree for g, so each root-to-leaf path is a rule of the form "if these counts increase, decrease, or stay the same, the transition is good", the same form as the rules of symbolic general policies over numerical features (Bonet & Geffner, 2018; Francès et al., 2021).

5 SMALL TRAINING INSTANCES COVER LARGE ONES

R-GNN policies are trained on small instances and used on much larger ones, and so are the concepts we decode from them. Does a larger instance contain cases the encoder never saw during training? We show that, for single objects, it does not: for a fixed domain and number of layers, a finite set of training states contains every computation the encoder performs on an object of any instance.

Why small instances suffice. The encoder computes an object's embedding only from what the object sees within L layers: its own atoms, its neighbors' atoms, their neighbors' atoms, and so on. Because messages are combined by a maximum, this view records which kinds of neighbors an object has, but not how many. We capture it by a weight-independent summary: the classical color refinement (1-WL) procedure used to study the power of GNNs (Morris et al., 2019; Xu et al., 2019), applied to atoms instead of edges. Two facts then give the result. First, objects with the same summary are treated exactly alike by the encoder, whatever its weights: they get the same messages and embedding at every layer. Second, there are only finitely many summaries, since we start with one and each layer creates only finitely many new ones. A larger instance has more objects but no new summaries, so a finite set of states containing every summary covers all instances.

We call an object that the encoder treats exactly like o, at every layer and for every choice of weights, a *twin* of o, and a training set *complete* if every object in every reachable state of every instance in $\mathcal{Q}$ has a twin in it (Definition 2 in Appendix A.2.1). Since twins are defined for all weights, completeness depends only on the training set and $\mathcal{Q}$.

Theorem 2 (Finite cover, proof in Appendix A.2.4)**.** *Fix a domain, a nonempty instance class $\mathcal{Q}$, and a number of layers L. Then, (a) there is a finite complete training set; and (b) there is a number N such that all reachable states of all instances in $\mathcal{Q}$ with at most N objects form a finite complete training set.*

What this means for decoding. Theorem 2 concerns the encoder only: the readout sums the final embeddings, and these sums can take new values as instances grow, so there is no guarantee for the readout or the neural policy. For decoding, it means that the embedding vectors to be explained form a finite set that already occurs in small instances, which supports decoding one concept per embedding vector (Theorem 1) and suggests that concepts consistent with the network on a complete training set remain consistent on larger instances. We do not prove the latter in general, and whether our samples are complete is an empirical question; in our experiments, no embedding vector unseen during training appears on the test instances (Appendix C.4).

Limits. The theorem relies on max aggregation, but not on binary embeddings (Appendix A.2.3). Twins receive the same *set* of messages, but not necessarily the same number of copies: an object in a large instance may receive a message many more times than its twin, which only the maximum makes irrelevant. The encoder therefore cannot distinguish objects that differ only in how many neighbors of some kind they have. Moreover, the theorem only says that a complete training set exists, not that random sampling finds one, that it is small, or how large N is.

Explicit bounds for Gripper and Ferry. For some domains, we can bound the instance size. Let $\mathcal{Q}_{grip}$ be the class of Gripper instances with two rooms, two grippers, any number of balls and a single `at` goal, with goal ball b^* and goal room r^*; every other ball is *ordinary*. Let $\mathcal{Q}_{ferry}$ be the class of Ferry instances with any number of locations and cars and a single `at` goal, with goal car c^* and goal location l^*; every other car is ordinary. For these classes, every feature vector (Section 4.1) that can occur already occurs in small instances.

Theorem 3 (Explicit bounds)**.** *For every layer $t \geq 1$: (a) every layer-t feature vector that occurs in any instance of $\mathcal{Q}_{grip}$ also occurs in an instance of $\mathcal{Q}_{grip}$ with at most three balls; and (b) every layer-t feature vector that occurs in any instance of $\mathcal{Q}_{ferry}$ also occurs in an instance of $\mathcal{Q}_{ferry}$ with at most three locations and three cars.*

The proofs are in Appendix A.2.5. For decoding, these small instances are thus as good as a complete training set: they contain every case the decoded concepts must handle.

6 EXPERIMENTS

We evaluate our claims on 12 planning domains, asking four questions. Do the decoded concepts reproduce the network, on training states and on much larger test instances? How small are the concepts, and how often are they certified minimal? Do the symbolic policies match the network? And are the policies readable and provably correct?

6.1 SETUP

Domains and models. We use 12 domains, each restricted to instances with a single goal atom and split by size into training, validation, and test instances; the test instances have up to two orders of magnitude more objects than the training instances (Table 3 in Appendix C.1). For each domain, we train R-GNN transition classifiers (Section 3) with $L = 2$ layers, embedding size $k = 16$, binary embeddings, and L1 regularization with $\lambda = 10^{-4}$, over 5 random seeds for 12 hours each on the training instances, using the validation instances to detect overfitting. Separate validation instances are not needed after training, so we also use them for decoding and rule learning.

Decoding and baselines. We decode each network from $40{,}000$ sampled states per domain, stopping the enumeration of prime implicants after $20{,}000$ per embedding vector (Section 4.2). We compare with three alternative decoders, all decoding one concept per embedding vector as ours does. **IDT** greedily fits each concept with a decision tree over our features (Section 4.1) and is inspired by prior work (Pluska et al., 2024). **Layered ALCSAT** applies the SAT-based concept learner ALCSAT (Funk et al., 2026) layer by layer, over the concepts of the previous layer. **Flat ALCSAT** uses the same learner to decode the last layer directly over the domain's predicates. Both ALCSAT variants search the full $\mathcal{ALCI}$ grammar, which is larger than our feature pool, and get 1 hour per concept, 2 hours for flat ALCSAT since it decodes only one layer. All decoders share the readout rules of Section 4.3, so only their concepts differ.

Policies. We run each policy on the test instances with a horizon of $1{,}000$ steps and cycle avoidance, moving in each state to the best successor not visited yet, using the model with the lowest validation loss among the 5 seeds. We report the number of solved instances (*coverage*) and the plan quality q, the mean ratio of plan length to optimal plan length over the solved instances.

6.2 RESULTS

Do the concepts reproduce the network? On the sampled states, our concepts are consistent with the network on every (state, object) pair in every domain (Table 4 in Appendix C.2), as Proposition 1 guarantees; no feature vector ever occurs with two different embeddings. IDT, which uses the same features, is also perfectly consistent. The ALCSAT variants are not: their search does not scale to the concept sizes some embeddings need, with mean consistency 0.965 (layered) and 0.935 (flat).

More important is whether the concepts still hold on the test instances, which are much larger than all instances used for decoding. Section 5 suggests that this depends on whether the sample is complete, i.e. whether the test instances produce embedding vectors the sample lacks. In our experiments, they never do: across all domains and test instances, no object receives an embedding vector unseen during decoding (Table 7 in Appendix C.4). The decoded concepts are exactly consistent with the network on the test instances of 9 of the 12 domains, and on at least 98.6% of the objects in the other three, where some test objects have feature vectors that never occur in the decoding sample.

How small are the concepts? Our concepts have a mean size of 13.1, $4.4\times$ smaller than IDT's 57.7 and smaller in every domain (Table 1). Both use the same features, so the difference lies in the search: IDT chooses features greedily, while we find a minimal DNF. For 73% of the concepts, including all first-layer concepts, the prime-implicant enumeration finishes within its budget, so they are certified minimal (Table 6 in Appendix C.3). Where the ALCSAT variants are perfectly consistent, as in `ferry`, `gripper` (for layered) and `satellite` (for both), their concepts are

Table 1: Mean size of the decoded concepts and number of solved test instances per domain, for the four decoders and for the R-GNN policy itself. The full tables, with consistency and plan quality, are in Appendix C.2. "n" is the number of test instances. Bold marks the best coverage per domain.

	mean concept size					test instances solved				
domain	ours	IDT	lay.	flat	n	R-GNN	ours	IDT	lay.	flat
`blocks-clear`	6.2	17.8	7.6	1.0	133	117	**133**	**133**	**133**	**133**
`blocks-on`	7.7	27.8	7.5	1.0	150	**150**	149	149	**150**	48
`blocks-on-table`	4.6	12.4	5.5	3.0	136	**136**	**136**	**136**	**136**	135
`childsnack`	7.3	55.7	3.7	5.6	164	124	124	104	**129**	124
`depots`	22.1	135.1	5.2	5.5	279	275	**279**	278	255	254
`driverlog`	23.6	67.1	3.1	3.5	112	**80**	75	79	76	69
`ferry`	6.1	13.3	5.5	4.1	156	**156**	**156**	**156**	**156**	**156**
`gripper`	4.2	16.0	3.0	4.1	117	**117**	**117**	**117**	**117**	**117**
`miconic`	9.3	20.5	2.0	1.0	104	**104**	**104**	**104**	11	29
`satellite`	3.6	5.2	1.9	1.8	144	**144**	**144**	**144**	**144**	**144**
`spanner`	8.5	18.0	3.0	2.5	90	**90**	**90**	**90**	0	0
`transport`	15.3	46.3	3.9	3.5	171	150	**155**	152	**155**	120
all 12 domains	13.1	57.7	4.4	4.6	1756	1643	**1662**	1642	1462	1329
coverage						93.6%	**94.6%**	93.5%	83.3%	75.7%

smaller than ours, since they are not restricted to DNFs over our features; elsewhere, their smaller concepts come at the price of lower consistency. Decoding per embedding dimension instead of per vector roughly doubles concept size (25.4 vs. 13.1; Appendix C.5).

L1 regularization gives simpler concepts. Stronger L1 regularization measurably simplifies the decoded concepts (Table 9 in Appendix C.6): from $\lambda = 10^{-5}$ to $\lambda = 10^{-4}$, their total size drops from $6{,}071$ to $5{,}027$ and the number of distinct embedding vectors from 459 to 383. With $\lambda = 10^{-3}$, however, the embeddings collapse to a single vector in seven domains, as the loss is dominated by the regularization term. For feedforward networks, sparsity penalties are known to give more concise rules (Setiono, 1997) and smaller approximating decision trees (Schaaf et al., 2019), although penalties designed for tree size do better (Wu et al., 2018); we are not aware of such measurements for relational GNNs, where the decoded concepts are exact. This suggests one possible reason why L1 regularization often helps generalization.

Do the symbolic policies match the network? The symbolic policies built from our concepts solve 94.6% of the test instances, on par with the 93.6% of the R-GNN policies (Table 1). Where they fail on some instances, as in `childsnack`, `driverlog`, and `transport`, the network fails on a similar number. In the five domains where the R-GNN policy is optimal ($q = 1.00$: `ferry`, `gripper`, `miconic`, `satellite`, and `spanner`), the symbolic policy is optimal too. Plan quality is otherwise mixed (Table 5 in Appendix C.2). Among the decoders, coverage does not separate ours from IDT (94.6% against 93.5%); concept size does. The ALCSAT variants solve clearly fewer instances (83.3% and 75.7%), mostly where they fail to fit the network.

6.3 Readable and provably correct policies: Gripper

In Gripper, a robot with two grippers moves balls between two rooms, and the goal is to move one ball to the other room. The symbolic policy tests only four layer-2 concepts, which read as follows (definitions in Appendix D):

- $C_1^{(2)}$: the goal room, while the goal ball is undelivered and the robot is elsewhere;
- $C_3^{(2)}$: a free gripper;
- $C_6^{(2)}$: the robot's room, if it is not the goal room and the undelivered goal ball does not lie in it;
- $C_8^{(2)}$: the goal ball, while undelivered, and the room it lies in.

Read as an ordered list, where the first matching rule decides, the rules are:

- r_1: if $\Delta_8 < 0$, good: pick up the goal ball, or drop it in the goal room;

r_2: otherwise, if $\Delta_1 > 0$ and $\Delta_6 \leq 0$, good: leave the goal room without the goal ball;
r_3: otherwise, if $\Delta_6 < 0$ and $\Delta_3 \leq 0$, good: carry the goal ball into the goal room;
f: otherwise, if $\Delta_1 \leq 0$, $\Delta_6 \geq 0$ and $\Delta_3 > 0$, fallback: drop a non-goal ball.

All other transitions are bad. The policy moves to a successor that r_1–r_3 accept, and to a fallback successor only if there is none; dropping a non-goal ball is optimal exactly when the other gripper also holds one. Appendix D shows the decision tree behind these rules. Since the concepts and rules are explicit, we can reason about the policy on every instance of the class, not only the test instances.

Theorem 4 (Gripper optimality, proof in Appendix D.1)**.** *On every instance of $\mathcal{Q}_{grip}$, the symbolic policy of Section 6.3 (rules r_1–r_3 and f, Table 11) over the concepts of Table 10 reaches the goal in exactly $V^*(s_0)$ steps.*

7 Related work

Learning general policies. The R-GNNs we decode come from work that learns general policies with GNNs, relates their power to the counting logic C_2, and raises the question of their transparency (Ståhlberg et al., 2022a;b; 2023); other neural approaches include Toyer et al. (2020); Rivlin et al. (2020); Chen et al. (2024). Symbolic approaches search grammar-generated pools of description-logic features under policy constraints (Bonet & Geffner, 2018; Francès et al., 2021; Drexler et al., 2022); we instead derive the features from a trained network.

Expressiveness and size generalization of GNNs. Aggregate-combine GNNs capture exactly graded modal logic, the modal reading of $\mathcal{ALCQ}$, and reach FOC_2 only with a global readout (Barceló et al., 2020; Morris et al., 2019; Xu et al., 2019; Grohe, 2021); Theorem 1 is an effective, per-network counterpart for max aggregation and binary embeddings. Yehudai et al. (2021) show that GNNs can fail on larger graphs with local structures that small ones lack; with max aggregation, all local structures the encoder distinguishes occur in instances up to some size (Section 5).

Extracting symbolic models from networks. Rule extraction and GNN explainers give approximate, post-hoc explanations (Andrews et al., 1995; Craven & Shavlik, 1996; Ying et al., 2019; Azzolin et al., 2023), while our concepts reproduce the network exactly on the sampled states. Closest to our work, Pluska et al. (2024) extract GNNs over a single binary relation into iterated decision trees; our IDT baseline takes inspiration from their method and adapts it to our features and networks, but our decoder also adds the guarantee that each concept is minimal. Description-logic concept learners, such as bounded fitting with SAT (ten Cate et al., 2023), ALCSAT (Funk et al., 2026) and DL-Learner (Lehmann et al., 2011), search all concepts up to a size bound, while we search only Boolean combinations of the features that one layer can compute. Other work builds interpretability into the model: Cucala et al. (2022; 2023) design GNNs from which Datalog programs can be extracted, and concept bottleneck models (Koh et al., 2020) and logic explained networks (Barbiero et al., 2022) build concepts into the model; we decode networks trained as usual, using only constraints that R-GNNs already have.

8 Conclusion

R-GNNs with max aggregation and binary embeddings are not black boxes. Every layer computes a Boolean function of a finite set of $\mathcal{ALCI}$ features derived from the network itself, so its embeddings can be decoded into description-logic concepts without designing or searching a feature pool. Our decoder finds these concepts exactly: they are consistent with the network on all sampled states, and 73% of them are certified to be the smallest such DNFs. Small instances suffice for this, since a finite set of training states contains every computation the encoder performs on an object of any instance; in our experiments, the much larger test instances never produce an embedding unseen during decoding. Replacing the network by the decoded concepts and rules over their counts yields symbolic policies that solve as many test instances as the network (94.6% against 93.6%), stay optimal wherever the network is, and, for Gripper, are provably optimal on every instance. L1 regularization also measurably simplifies the learned concepts. Together, these results suggest that R-GNNs generalize because they learn logical features that support general policies, and that these features can be made explicit.

AI USE STATEMENT

In this work, we used generative AI tools, in the form of large language model assistants, for the following tasks with required disclosure: to implement methods, by writing and refactoring parts of the code and analysis scripts; to assist in the writing of proofs; to provide feedback on the design of experiments; and to help interpret results. We have not used generative AI tools to develop the theoretical framework, to formulate the mathematical claims, or to propose the hypotheses of the paper, which come from the authors; generating synthetic datasets, translation, cleaning or reformatting datasets, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for tasks with recommended disclosure: to aggregate results into tables and figures, to brainstorm and discuss research ideas and related work, to identify relevant literature, and to edit the text of the paper for readability. We have reviewed all AI-assisted work: all AI-generated code was reviewed and tested by the authors, every number reported in the paper was produced by running the code, to be released upon acceptance, and all AI-assisted text and proofs were checked by the authors for correctness. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only standard planning benchmarks generated from public domain descriptions; it involves no human subjects and no personal or sensitive data. Its aim is to make learned policies more transparent by turning them into symbolic policies that can be read and, in some cases, proven correct. We do not see direct risks of harm or misuse beyond those of automated planning in general.

REPRODUCIBILITY STATEMENT

All code, trained models, and data will be released upon acceptance, together with the scripts and configurations used for every experiment. The architecture and training of the networks are described in Section 3, and their hyperparameters in Section 6.1. The decoder is specified in Section 4, with the SAT and MaxSAT encodings and the enumeration budget in Appendix B and a worked example in Appendix B.1. The benchmark splits are in Appendix C.1, and all proofs in Appendices A and D.1.

REFERENCES

Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Liò, and Andrea Passerini. Global explainability of GNNs via logic combination of learned concepts. In *11th International Conference on Learning Representations (ICLR)*, 2023.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (eds.). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

Aniket Nick Bajpai, Sankalp Garg, et al. Transfer of deep reactive policies for mdp planning. In *Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS 2018)*, pp. 10965–10975, 2018.

Pietro Barbiero, Gabriele Ciravegna, Francesco Giannini, Pietro Lió, Marco Gori, and Stefano Melacci. Entropy-based logic explanations of neural networks. In Vasant Honavar and Matthijs Spaan (eds.), *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, pp. 6046–6054. AAAI Press, 2022.

Pablo Barceló, Egor Kostylev, Mikaël Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*. OpenReview.net, 2020.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. In Jérôme Lang (ed.), *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pp. 4667–4673. IJCAI, 2018.

Craig Boutilier, Raymond Reiter, and Bob Price. Symbolic dynamic programming for first-order MDPs. In Bernhard Nebel (ed.), *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pp. 690–700. Morgan Kaufmann, 2001.

Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

Tathagata Chakraborti, Anagha Kulkarni, Sarath Sreedharan, David E. Smith, and Subbarao Kambhampati. Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In Robert P. Goldman, Susanne Biundo, and Michael Katz (eds.), *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, pp. 86–96. AAAI Press, 2021.

Dillon Z. Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In Jennifer Dy and Sriraam Natarajan (eds.), *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, pp. 20078–20086. AAAI Press, 2024.

Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems 8 (NIPS)*, pp. 24–30. MIT Press, 1996.

David J. Tena Cucala, Bernardo Cuenca Grau, Egor V. Kostylev, and Boris Motik. Explainable GNN-based models over knowledge graphs. In *10th International Conference on Learning Representations (ICLR)*, 2022.

David J. Tena Cucala, Bernardo Cuenca Grau, Boris Motik, and Egor V. Kostylev. On the correspondence between monotonic max-sum GNNs and Datalog. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2023.

Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In Sylvie Thiébaux and William Yeoh (eds.), *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pp. 62–70. AAAI Press, 2022.

Alan Fern, SungWook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

Guillem Francès, Blai Bonet, and Hector Geffner. Learning general planning policies from small examples without supervision. In Kevin Leyton-Brown and Mausam (eds.), *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pp. 11801–11808. AAAI Press, 2021.

Maurice Funk, Jean Christoph Jung, and Tom Völlmer. Bounded fitting for expressive description logics. In *35th International Joint Conference on Artificial Intelligence (IJCAI 2026)*, 2026.

Martin Grohe. The logic of graph neural networks. In *Proceedings of the Thirty-Sixth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*, pp. 1–17, 2021.

Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In Toby Walsh (ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 918–923. AAAI Press, 2011.

Sergio Jiménez, Javier Segovia-Aguas, and Anders Jonsson. A review of generalized planning. *The Knowledge Engineering Review*, 34:e5, 2019.

Roni Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, pp. 5338–5348. JMLR.org, 2020.

Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9(1):71–81, 2011.

Mario Martín and Hector Geffner. Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, 20(1):9–19, 2004.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pp. 4602–4609, 2019.

Alexander Pluska, Pascal Welke, Thomas Gärtner, and Sagar Malhotra. Logical distillation of graph neural networks. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*, 2024.

Or Rivlin, Tamir Hazan, and Erez Karpas. Generalized planning with deep reinforcement learning. In *ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, pp. 16–24, 2020.

Scott Sanner and Craig Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5-6):748–788, 2009.

Nina Schaaf, Marco F. Huber, and Johannes Maucher. Enhancing decision tree based interpretation of deep neural networks through L1-orthogonal regularization. In *Proceedings of the 18th IEEE International Conference on Machine Learning and Applications (ICMLA 2019)*, pp. 42–49, 2019. doi: 10.1109/ICMLA.2019.00016.

Rudy Setiono. A penalty-function approach for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204, 1997. doi: 10.1162/neco.1997.9.1.185.

Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):393–401, 2011.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In Sylvie Thiébaux and William Yeoh (eds.), *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pp. 629–637. AAAI Press, 2022a.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning generalized policies without supervision using GNNs. In Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer (eds.), *Proceedings of the Nineteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2022)*, pp. 474–483. IJCAI Organization, 2022b.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner (eds.), *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, pp. 647–657. IJCAI Organization, 2023.

Balder ten Cate, Maurice Funk, Jean Christoph Jung, and Carsten Lutz. SAT-based PAC learning of description logic concepts. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3347–3355, 2023.

Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. ASNets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68:1–68, 2020.

Mike Wu, Michael Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11501.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the Seventh International Conference on Learning Representations (ICLR 2019)*. OpenReview.net, 2019.

Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11975–11986, 2021.

Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNNExplainer: Generating explanations for graph neural networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

A PROOFS

This appendix gives the proofs for the theorems and the proposition of the paper, except the proof of Theorem 4, which is in Appendix D together with the Gripper policy. Appendix A.1 proves Theorem 1 and Proposition 1. Appendix A.2 proves the results of Section 5. There, we first state the setting in full (Appendix A.2.1) and define colors, the weight-free summaries described informally in Section 5 (Appendix A.2.2). We then prove the key lemma: equal colors give equal network computations (Appendix A.2.3), and the finite cover theorem follows by counting colors (Appendix A.2.4). Finally, we prove the explicit bounds for Gripper and Ferry (Appendix A.2.5).

We write $\mathbf{f}^{(t)}(o) \in \{0,1\}^{|\mathcal{L}^{(t)}|}$ for the feature vector of an object o at layer t (Section 4.1), which is used in the proofs of Theorem 1 and Proposition 1.

A.1 PROOFS FOR SECTION 4

A.1.1 PROOF OF THEOREM 1

Theorem 1 (Sufficiency of the feature pool)**.** *For domains whose predicates have arity at most two, the features $\mathcal{L}^{(t)}$ determine the embeddings of layer t. That is, let o and o' be objects of states whose layer-$(t-1)$ embeddings lie in $X^{(t-1)}$; if every feature in $\mathcal{L}^{(t)}$ has the same truth value at o and at o', then $\mathbf{x}^{(t)}(o) = \mathbf{x}^{(t)}(o')$.*

Proof. Fix a layer t and an object o of a state whose layer-$(t-1)$ embeddings lie in $X^{(t-1)}$. By Equation 3, $\mathbf{x}^{(t)}(o)$ is a function of the pair $\big(\mathbf{x}^{(t-1)}(o), m^{(t)}(o)\big)$. The first component is determined by the feature vector: exactly one of the features C_x, $x \in X^{(t-1)}$, is true at o, namely the one with $x = \mathbf{x}^{(t-1)}(o)$. For the second component, recall from Equation 2 that $m^{(t)}(o)$ is the componentwise maximum of the set $S^{(t)}(o)$ of messages that o receives, so it suffices to show that $S^{(t)}(o)$ is determined by the feature vector. By Equation 1, the message that an atom sends to o depends only on the predicate, the position of o in the atom, and the layer-$(t-1)$ embeddings of the atom's arguments. A unary atom $p(o)$ sends a message that depends only on p and on $\mathbf{x}^{(t-1)}(o)$, and whether it is sent is the truth value of the feature p at o. A binary atom $p(o, \bar{o})$ sends to o a message that depends only on p, on $\mathbf{x}^{(t-1)}(o)$, and on $\mathbf{x}^{(t-1)}(\bar{o}) \in X^{(t-1)}$; such a message is sent exactly when o has a p-neighbor with embedding $\mathbf{x}^{(t-1)}(\bar{o})$, which is the truth value of the feature $\exists p.C_{\mathbf{x}^{(t-1)}(\bar{o})}$ at o. The same holds for an atom $p(\bar{o}, o)$ with the inverse role p^-. The features only include roles and their inverses, which is the reason for the restriction to predicates of arity at most two. Since $S^{(t)}(o)$ is a set, it does not matter how many neighbors send the same message. Hence $S^{(t)}(o)$, and therefore $\mathbf{x}^{(t)}(o)$, is determined by $\mathbf{f}^{(t)}(o)$, so two objects with the same feature vector have the same embedding at layer t. □

A.1.2 PROOF OF PROPOSITION 1

Proposition 1 (Consistency and minimality)**.** *Suppose that the concepts of layer $t-1$ are consistent with the network. Then every concept C_x of layer t is consistent with the network. If, in addition, all prime implicants are found, then the total size of the conjunctions of C_x is the smallest among all DNFs over $\mathcal{L}^{(t)}$ that are consistent with the network.*

Proof. Since the concepts of layer $t-1$ are consistent with the network, each feature $C_{x'}$ and $\exists r.C_{x'}$ has, on every sampled object, the truth value it has under the reading of Section 4.1. So the feature vector computed with the decoded concepts is $\mathbf{f}^{(t)}(o)$ for every sampled object o. By construction, C_x is a disjunction of implicants, which are false on every negative example, and the chosen implicants together are true on every positive example. A sampled object o with $\mathbf{x}^{(t)}(o) = x$ has a positive feature vector, so $o \in [\![C_x]\!]_s$. A sampled object with a different embedding has a negative feature vector, so $o \notin [\![C_x]\!]_s$. Hence C_x is consistent with the network.

For minimality, let F be a DNF over $\mathcal{L}^{(t)}$ that is consistent with the network, with the smallest total size of its conjunctions, and let c be a conjunction of F. If c is true on no positive example, it can be dropped. Otherwise, c is false on every negative example, since F is, so c is an implicant. Dropping

literals from c until no more can be dropped gives a prime implicant c' that is true wherever c is true and satisfies $|c'| \leq |c|$, since removing literals never increases the size. Replacing every conjunction of F in this way gives a set of prime implicants that is true on every positive example and has a total size of at most that of F. If all prime implicants are found, this set is one of the candidates of the cover step, which returns a cheapest candidate, so the total size of the conjunctions of C_x is at most that of F. □

A.2 Proofs for Section 5

A.2.1 Setting recap

Here we restate the parts of Sections 2 and 3 that the proofs rely on.

Instances and states. Every instance P has a finite, nonempty set of objects $\mathcal{O}_P$; ground atoms may repeat arguments, and each instance has only finitely many states, reachable or not. Object names do not matter: renaming objects in an instance and its state changes neither colors nor embeddings, so when counting instances with n objects we use the names $\{1, \ldots, n\}$ and count renamed copies once.

Input. The encoder reads the atoms $\mathrm{Input}(s, g_P)$ of Section 3: the atoms of s, plus one atom $p_{g,t}$ or $p_{g,f}$ for each goal atom. The proofs only use that $\mathrm{Input}(s, g_P)$ is determined by s and g_P, uses a fixed finite set of predicates $\mathcal{P}^*$ of fixed positive arities κ_p, and is renamed along with the objects; any other input with these properties works as well. An atom is *incident* to its arguments and sends one message to each argument position, so an object occurring in several positions of an atom receives one message per position.

The encoder. The proof holds for a more general encoder than the one of the R-GNN (Section 3). Fix an embedding dimension $k \geq 1$, a number of layers $L \geq 1$, and weights θ, and let $B = \{0,1\}^k$. Every object starts with the same vector $b_0 \in B$:

$$\mathbf{x}^{(0)}_{\theta,P,s}(o) = b_0.$$

At layer $t \in \{1, \ldots, L\}$, messages have a fixed dimension $d_t \geq 1$. For each predicate $p \in \mathcal{P}^*$ and position $i \in \{1, \ldots, \kappa_p\}$, there is a message function $M_{\theta,t,p,i} : B^{\kappa_p} \to \mathbb{R}^{d_t}$. An atom $p(o_1, \ldots, o_{\kappa_p}) \in \mathrm{Input}(s, g_P)$ sends to its argument in position i the message

$$M_{\theta,t,p,i}\big(\mathbf{x}^{(t-1)}_{\theta,P,s}(o_1), \ldots, \mathbf{x}^{(t-1)}_{\theta,P,s}(o_{\kappa_p})\big).$$

The set $S^{(t)}_{\theta,P,s}(o)$ contains every message vector sent to o at layer t, with repeated vectors listed once. The aggregate $m^{(t)}_{\theta,P,s}(o)$ is the componentwise maximum of this set. If the set is empty, the aggregate is a fixed vector $e_t \in \mathbb{R}^{d_t}$, the same for all objects, states and instances. Finally, an update function $U_{\theta,t} : B \times \mathbb{R}^{d_t} \to B$ gives the next embedding:

$$\mathbf{x}^{(t)}_{\theta,P,s}(o) = U_{\theta,t}\big(\mathbf{x}^{(t-1)}_{\theta,P,s}(o),\ m^{(t)}_{\theta,P,s}(o)\big).$$

All message and update functions are deterministic and are shared across objects, states, and instances. They may also be shared across layers. They use only the inputs shown above, and no other inputs such as object names, object counts, or degrees. Any binarization is part of the update, and the messages may be real-valued.

The encoder of the R-GNN (Section 3) is a special case. It uses $b_0 = \mathbf{0}$; the message function $M_{\theta,t,p,i}$ is the i-th output of MLP_p in Equation 1; the aggregate is the maximum of Equation 2; and the update is $U_{\theta,t}(\mathbf{x}, m) = \mathrm{bin}\big(\mathrm{MLP}_U(\mathbf{x}, m)\big)$ from Equation 3. Its message sets are never empty, because of the atoms $\mathsf{object}(o)$, so the default aggregate e_t is not needed there.

Twins and complete training sets. Section 5 uses the following notions.

Definition 2 (Twins and completeness)**.** An object o' in state s' of instance P' is a *twin* of an object o in state s of instance P if, for every choice of weights θ,

$$\mathbf{x}^{(t)}_{\theta,P,s}(o) = \mathbf{x}^{(t)}_{\theta,P',s'}(o') \qquad (0 \leq t \leq L),$$

$$S^{(t)}_{\theta,P,s}(o) = S^{(t)}_{\theta,P',s'}(o') \quad \text{and} \quad m^{(t)}_{\theta,P,s}(o) = m^{(t)}_{\theta,P',s'}(o') \qquad (1 \leq t \leq L).$$

A training set T is *complete* if every object o in every reachable state s of every instance $P \in \mathcal{Q}$ has a twin in some example $(P', s') \in T$.

A.2.2 COLORS

Colors make precise the summary of "what an object could know after t layers" from Section 5. In each round, an object's new color collects its old color and the old colors of the objects it shares an atom with. Colors are computed from the states alone, without any weights.

Definition 3 (Colors)**.** Let (P, s) be a training example and $o \in \mathcal{O}_P$. Initially, every object has the same color, $c^{(0)}_{P,s}(o) = *$. For round $t \geq 0$, each atom $p(o_1, \ldots, o_{\kappa_p}) \in \mathrm{Input}(s, g_P)$ and each position i with $o_i = o$ give the *pattern*

$$\big(p,\ i,\ c^{(t)}_{P,s}(o_1),\ \ldots,\ c^{(t)}_{P,s}(o_{\kappa_p})\big).$$

Let $\Pi^{(t)}_{P,s}(o)$ be the set of all such patterns, with repeated patterns listed once. The new color is the pair

$$c^{(t+1)}_{P,s}(o) = \big(c^{(t)}_{P,s}(o),\ \Pi^{(t)}_{P,s}(o)\big).$$

We write $\mathcal{K}^{(t)}$ for the set of colors that occur after t rounds on objects of reachable states of instances in $\mathcal{Q}$.

A pattern says "I am in position i of a p-atom, and the arguments of this atom had these colors". A color is just this record, so two objects get the same new color exactly when they had the same old color and the same set of patterns. Different records are always different colors. This is how colors are defined; it is not an assumption about the network, which may give the same embedding to different colors. Information from farther away arrives through the old colors: a neighbor's color after one round already describes the neighbor's own atoms, so after t rounds a color summarizes everything within t steps of the object. Patterns form a *set*. If o occurs in $r(o, a)$ and $r(o, b)$, and a and b have the same color, the two atoms give the same pattern, and it is recorded once. This matches the maximum in Equation 2, which also ignores how many copies of a message arrive.

We say that a training set *covers all colors* if every color in $\mathcal{K}^{(L)}$ occurs on some object of some training state. Since a color contains all of the object's earlier colors, such a set also covers every color of every earlier round.

A.2.3 EQUAL COLORS GIVE EQUAL COMPUTATIONS

The first step shows that the network cannot separate objects that the coloring does not separate. The idea is that the coloring records everything the network uses: at each layer, the network combines an object's own previous embedding with messages from incident atoms, and each message depends only on the predicate, the position, and the previous embeddings of the atom's arguments. The coloring records exactly these ingredients, with previous colors in place of previous embeddings.

Lemma 1 (Equal colors give equal computations)**.** *Fix any weights θ and any $t \in \{0, \ldots, L\}$. Let o be an object in state s of instance P, and o' an object in state s' of instance P', with $c^{(t)}_{P,s}(o) = c^{(t)}_{P',s'}(o')$. Then $\mathbf{x}^{(t)}_{\theta,P,s}(o) = \mathbf{x}^{(t)}_{\theta,P',s'}(o')$. If $t \geq 1$, then also $S^{(t)}_{\theta,P,s}(o) = S^{(t)}_{\theta,P',s'}(o')$ and $m^{(t)}_{\theta,P,s}(o) = m^{(t)}_{\theta,P',s'}(o')$. Consequently, there is a function $f_{\theta,t} : \mathcal{K}^{(t)} \to B$ such that*

$$\mathbf{x}^{(t)}_{\theta,P,s}(o) = f_{\theta,t}\big(c^{(t)}_{P,s}(o)\big)$$

for every object o of every reachable state s of every instance $P \in \mathcal{Q}$.

So, after t layers, the embedding of an object is a function of its color after t rounds. The converse does not hold: different colors may map to the same embedding, and two objects with equal embeddings need not have equal colors, equal previous embeddings, or equal incoming messages. Theorem 1 makes a similar point, with the previous layer's embeddings in place of the previous colors. The difference is that colors do not depend on the weights, which is what allows the finite-cover theorem to choose the training set before training.

The proof goes one round at a time. The step from one round to the next is the argument from the paragraph before the lemma: equal patterns give equal messages, equal messages give equal aggregates, and equal aggregates give equal embeddings.

Proof. We use induction on t. For $t = 0$, every object has the embedding b_0, so the claim holds.

Now let $t \geq 1$ and assume that the claim holds for $t-1$. Suppose $c^{(t)}_{P,s}(o) = c^{(t)}_{P',s'}(o')$. By Definition 3, the two objects had the same previous color and the same set of patterns:

$$c^{(t-1)}_{P,s}(o) = c^{(t-1)}_{P',s'}(o'), \qquad \Pi^{(t-1)}_{P,s}(o) = \Pi^{(t-1)}_{P',s'}(o').$$

By the claim for $t-1$, the two objects have equal previous embeddings.

Take any message that o receives at layer t. It is sent by some atom $p(o_1, \ldots, o_{\kappa_p}) \in \mathrm{Input}(s, g_P)$ with $o_i = o$, and this atom gives the pattern $(p, i, c^{(t-1)}_{P,s}(o_1), \ldots, c^{(t-1)}_{P,s}(o_{\kappa_p}))$ in $\Pi^{(t-1)}_{P,s}(o)$. The same pattern is in $\Pi^{(t-1)}_{P',s'}(o')$, so there is an atom $p(o'_1, \ldots, o'_{\kappa_p}) \in \mathrm{Input}(s', g_{P'})$ with $o'_i = o'$ and $c^{(t-1)}_{P',s'}(o'_j) = c^{(t-1)}_{P,s}(o_j)$ for every j. By the claim for $t-1$, the corresponding arguments o_j and o'_j have equal previous embeddings. The message function $M_{\theta,t,p,i}$ therefore receives equal inputs for the two atoms and sends the same vector to o and to o'.

Hence every vector in $S^{(t)}_{\theta,P,s}(o)$ is also in $S^{(t)}_{\theta,P',s'}(o')$. The same argument with the roles of o and o' swapped gives the other inclusion, so the two message sets are equal. Equal sets have equal componentwise maxima, and if both sets are empty, both aggregates are e_t. So $m^{(t)}_{\theta,P,s}(o) = m^{(t)}_{\theta,P',s'}(o')$. The update now receives equal previous embeddings and equal aggregates, so it returns equal embeddings. This proves the claim for t.

For the function $f_{\theta,t}$, let $\gamma \in \mathcal{K}^{(t)}$. By the definition of $\mathcal{K}^{(t)}$, some object has color γ; let $f_{\theta,t}(\gamma)$ be its embedding after t layers. By the claim, every object with color γ has this same embedding, so $f_{\theta,t}$ is well defined. □

The proof uses the maximum only through one fact: the aggregate depends only on the *set* of incoming messages. Any other aggregation with this property, such as a componentwise minimum, would work as well. A sum or a mean would not, since it also depends on how many copies of each message arrive. The proof also never uses that the embeddings are binary. It holds for real-valued embeddings under the same constant start and the same local, deterministic updates.

A.2.4 Proof of Theorem 2

The key statement is that a finite training set contains every color: for every object with color c in a reachable state of an instance in $\mathcal{Q}$, some object in a state of the training set also has color c. Since equal colors give equal computations, the network then sees every local computation during training. The proof has three steps. First, we count colors and show that there are only finitely many in each round. Second, we show that every training set that covers all colors is complete, since an object with the same final color is a twin by Lemma 1. Picking one training example for each final color then gives part (a). Third, we take N to be the largest number of objects among the picked examples, which gives part (b).

Theorem 2 (Finite cover, proof in Appendix A.2.4)**.** *Fix a domain, a nonempty instance class $\mathcal{Q}$, and a number of layers L. Then, (a) there is a finite complete training set; and (b) there is a number N such that all reachable states of all instances in $\mathcal{Q}$ with at most N objects form a finite complete training set.*

Proof. There are only finitely many colors in each round. Let $n_t = |\mathcal{K}^{(t)}|$. Initially $n_0 = 1$. Suppose n_t is finite. A pattern for predicate p has κ_p possible positions and at most $n_t^{\kappa_p}$ ordered lists of argument colors, so the number of possible patterns is at most

$$q_t = \sum_{p \in \mathcal{P}^*} \kappa_p \, n_t^{\kappa_p}.$$

A set of patterns is a subset of these possible patterns, so there are at most 2^{q_t} such sets. A new color consists of one old color and one set of patterns, so

$$n_{t+1} \leq n_t \, 2^{q_t} < \infty.$$

Some of the counted combinations may never occur, which only makes n_{t+1} smaller. By induction, n_L is finite.

Covering all colors implies completeness. Let T be any training set that covers all colors, and let o be an object in a state $s \in \mathrm{Reach}(P)$ of an instance $P \in \mathcal{Q}$. Its color $c^{(L)}_{P,s}(o)$ is in $\mathcal{K}^{(L)}$, so some object o' in some example $(P', s') \in T$ has the same color. Since a color contains all earlier colors, the two objects also have equal colors in every round $t \leq L$. Lemma 1 then gives equal embeddings for $0 \leq t \leq L$, and equal message sets and aggregates for $1 \leq t \leq L$, for every choice of θ. So o' is a twin of o, and T is complete.

Part (a). For each color $\gamma \in \mathcal{K}^{(L)}$, pick an instance $P' \in \mathcal{Q}$, a state $s' \in \mathrm{Reach}(P')$, and an object of P' that has color γ in s'. Such a choice exists because $\mathcal{K}^{(L)}$ contains only colors that occur. Let T be the set of the picked pairs (P', s'). It has at most n_L elements, so it is a finite training set. It covers all colors by construction, so it is complete by the previous step.

Part (b). Let T be the finite set from part (a), and let

$$N = \max\bigl\{|\mathcal{O}_{P'}| : (P', s') \in T\bigr\}.$$

This maximum exists because T is finite and nonempty ($\mathcal{Q}$ is nonempty, and each instance has at least its initial state and one object). For each $n \leq N$, name the objects $\{1, \ldots, n\}$. There are finitely many ground atoms over these objects, so there are finitely many possible initial states and goals, and hence finitely many instances with at most N objects. Each of them has finitely many reachable states. Taking all such instances that belong to $\mathcal{Q}$, together with all their reachable states, thus gives a finite training set. It contains every example of T, up to renaming of objects, and renaming does not change colors. So this training set also covers all colors, and it is complete by the second step. □

The proof shows more than the theorem states: *every* training set that covers all colors is complete. Unlike completeness, covering all colors is a property of the states alone, so it can be computed without the network. For example, one can count the colors that occur in a set of larger instances but not in the training set.

The proof picks only one example per color, so a complete training set with at most n_L states exists. But the bound on n_L grows very fast with L, and the proof gives no bound on N: it says nothing about how many objects are needed to produce a given color in a reachable state. The proof shows that n_L and N are finite, but it does not tell us their values. The network has at most 2^k embedding vectors, but there may be many more colors, since each choice of weights may give the same embedding to different colors.

A.2.5 Proof of Theorem 3

Theorem 3 (Explicit bounds)**.** *For every layer $t \geq 1$: (a) every layer-t feature vector that occurs in any instance of $\mathcal{Q}_{grip}$ also occurs in an instance of $\mathcal{Q}_{grip}$ with at most three balls; and (b) every layer-t feature vector that occurs in any instance of $\mathcal{Q}_{ferry}$ also occurs in an instance of $\mathcal{Q}_{ferry}$ with at most three locations and three cars.*

Proof. The layer-t feature vector of an object o is determined by its own layer-$(t-1)$ embedding and the set of pairs of a role and a layer-$(t-1)$ embedding of a neighbor along that role. By Lemma 1, these embeddings are functions of the round-$(t-1)$ colors, which the round-t color of o contains, so the feature vector is a function of the round-t color of o. It thus suffices to show that every color of an object in $\mathcal{Q}_{grip}$ or $\mathcal{Q}_{ferry}$ also occurs in a small instance of the same class.

Call two objects *linked* if they occur together in a binary atom of $\mathrm{Input}(s, g)$, including the goal atom, and consider the connected components of this relation. Nullary atoms, such as $\mathsf{empty-ferry}$, have no arguments and send no messages. By induction on the rounds, the color of o depends only on the atoms whose arguments lie in the component of o. Two operations therefore preserve the color of o. First, the part of the state outside the component of o can be replaced by any other part that is not linked to it. Second, if two objects a and a' are *structural twins*, i.e. swapping them maps $\mathrm{Input}(s, g)$ onto itself and no atom contains both, then deleting a' and its atoms preserves the colors of all other objects. To see this, note that a and a' have the same color in every round, by symmetry, and that every deleted atom $p(\ldots, a', \ldots)$ gives each of its other arguments the same pattern as the atom $p(\ldots, a, \ldots)$ obtained by the swap, which is kept; since patterns form a set, no pattern is lost.

In Gripper, the binary atoms are $\mathsf{at}(b, r)$, $\mathsf{carry}(b, h)$ and the goal atom $\mathsf{at}(b^*, r^*)$. So a component is a room with the balls lying in it, or a gripper with the ball it carries, where the goal atom joins the

component of b^* with that of r^*. Two ordinary balls in the same room are structural twins, so we delete all but one of them. The component of o then contains b^*, at most one ordinary ball in r^* and at most one in the other room, so at most three balls; if it does not contain b^*, it contains at most one ball. We complete it to a legal state of an instance in $\mathcal{Q}_{grip}$: we add the missing rooms and grippers, keep the robot where it is if its room is in the component and otherwise place it in an added room, and, if b^* is not in the component, let it be carried by an added gripper or lie in r^*; all other grippers are free. The result has two rooms, two grippers and at most three balls.

In Ferry, the benchmark uses the variant without the static predicate $\mathsf{not-eq}$: the ferry may sail to any location where it is not. The binary atoms are $\mathsf{at}(c, l)$ and the goal atom on (c^*, l^*), so a component is a location with the cars at it, or a single car on the ferry, where the goal atom again joins the components of c^* and l^*. Two ordinary cars at the same location are structural twins, so we delete all but one of them. The component of o then contains at most two locations and three cars, or, without c^*, at most one location and one car. We complete it as before: we add c^* and l^* if they are missing, placing c^* on the ferry or at l^*, and, if the ferry's location is not in the component, one location without cars for it; a car outside the component that is on the ferry is deleted and $\mathsf{empty-ferry}$ is set accordingly. The result has at most three locations and three cars.

In both cases, the completed state is a legal state of an instance in the class, and the classes place no restriction on initial states, so it is the (reachable) initial state of such an instance. □

B Implementation of the decoder

This appendix describes how we solve the decoding problem of Section 4 in practice.

Distinct feature vectors. We sample states, run the network on each of them, and compute for every object o and layer t the pair $\big(\mathbf{f}^{(t)}(o), \mathbf{x}^{(t)}(o)\big)$ of its feature vector and its embedding vector, where $\mathbf{f}^{(t)}(o) \in \{0,1\}^{|\mathcal{L}^{(t)}|}$ contains the truth values of the features of layer t at o. Since only these pairs matter (Theorem 1), we store each distinct pair only once. This gives a table with one row per distinct feature vector, and because planning states are highly constrained, it is small: many sampled objects share the same feature vector. If the same feature vector occurs with two different embedding vectors, then Theorem 1 does not hold for the network at hand, so we count such *collisions* and report them; in all our runs there are none.

Prime implicants as minimal hitting sets. Fix a layer t and a vector $x \in X^{(t)}$, and let E^+ and E^- be the distinct positive and negative feature vectors. For a positive vector $r \in E^+$ and a negative vector $u \in E^-$, the *difference set*

$$D_u^r = \{\ell \in \mathcal{L}^{(t)} : r[\ell] \neq u[\ell]\}$$

contains the features on which they differ. A conjunction of literals that is true on r uses each feature ℓ with the polarity $r[\ell]$, and it is false on u exactly when it contains a feature of D_u^r. The implicants that are true on r therefore correspond to the hitting sets of $\{D_u^r : u \in E^-\}$, and the prime implicants that are true on r to the minimal hitting sets. We encode the hitting-set condition as the formula

$$\Phi_r \;=\; \bigwedge_{u \in E^-} \; \bigvee_{\ell \in D_u^r} x_\ell, \tag{5}$$

where x_ℓ states that feature ℓ is used. Difference sets that are supersets of others can be dropped, since hitting the smaller set also hits the larger one. We enumerate all minimal models $\mathcal{I}_r$ of Φ_r with a SAT solver, adding one blocking clause per model until the solver returns UNSAT. We stop the enumeration after $20{,}000$ prime implicants per vector; a concept for which this happens is still consistent with the network, but it is not certified to be minimal (Appendix C.3).

Cheapest cover with MaxSAT. Let $\mathcal{I} = \bigcup_{r \in E^+} \mathcal{I}_r$ be all prime implicants found for x. We choose a cheapest set of them that covers every positive vector with weighted MaxSAT:

$$\text{hard:} \quad \bigwedge_{r \in E^+} \; \bigvee_{c \,\in\, \mathcal{I}_r} y_c, \tag{6}$$

$$\text{soft:} \quad \big\{(\neg y_c,\ |c|) \;:\; c \in \mathcal{I}\big\}, \tag{7}$$

where y_c states that c is chosen and $|c|$ is its size. The hard clauses ensure that every positive vector is covered by a chosen conjunction, and the soft clauses minimize the total size of the chosen conjunctions. The size matters because different features can describe the same objects at different cost; in blocks world, for example, $\exists \mathsf{on}^{-}.\top$ has size 2 and clear has size 1, while describing the same thing.

Rules for the readout. For the classifier g of Section 4.3, we train a decision tree (the `DecisionTreeClassifier` of scikit-learn) on the effect patterns of sampled transitions, labeled by the network. Each root-to-leaf path is a conjunction of sign tests, and we read it as a rule; Appendix D shows the tree for Gripper.

B.1 Worked example: blocks-clear

We decode one embedding vector of the first layer of the `blocks-clear` network, following Appendix B. Table 2 shows six of the ten distinct feature vectors of the first layer, together with the embedding vectors the network assigns to them. At layer 1, all objects have the same previous embedding $\mathbf{0}$, so $C_{\mathbf{0}} = \top$, and the features reduce to the unary predicates and to whether an object has any neighbor along each role.

Table 2: Six of the ten distinct layer-1 feature vectors of `blocks-clear`, with the embedding vector each receives.

	clear	$\mathsf{clear}_{g,t}$	$\mathsf{clear}_{g,f}$	$\mathsf{holding}$	$\mathsf{on\text{-}table}$	$\exists \mathsf{on}.\top$	$\exists \mathsf{on}^{-}.\top$	embedding
u_1	1	0	0	0	1	0	0	$x_1^{(1)}$
u_2	1	1	0	0	1	0	0	$x_3^{(1)}$
u_3	0	0	0	1	0	0	0	$x_4^{(1)}$
u_4	1	0	0	0	0	1	0	$x_2^{(1)}$
u_5	0	0	1	0	1	0	1	$x_6^{(1)}$
u_6	0	0	0	0	1	0	1	$x_3^{(1)}$

Difference sets. We decode $x_1^{(1)}$. Its only positive feature vector is u_1, and the other nine distinct feature vectors $u_2, \ldots, u_{10}$ are negative, since they receive different embeddings. This gives nine difference sets. Five of them are supersets of the following four

$$D_{u_2}^{u_1} = \{\mathsf{clear}_{g,t}\}, \qquad D_{u_3}^{u_1} = \{\mathsf{clear}, \mathsf{holding}, \mathsf{on\text{-}table}\},$$
$$D_{u_4}^{u_1} = \{\mathsf{on\text{-}table}, \exists \mathsf{on}.\top\}, \qquad D_{u_6}^{u_1} = \{\mathsf{clear}, \exists \mathsf{on}^{-}.\top\}.$$

Prime implicants. The hitting-set formula of Equation 5 is

$$\Phi_{u_1} = \big(x_{\mathsf{clear}_{g,t}}\big) \wedge \big(x_{\mathsf{clear}} \vee x_{\mathsf{holding}} \vee x_{\mathsf{on\text{-}table}}\big) \wedge \big(x_{\mathsf{on\text{-}table}} \vee x_{\exists \mathsf{on}.\top}\big) \wedge \big(x_{\mathsf{clear}} \vee x_{\exists \mathsf{on}^{-}.\top}\big),$$

which has four minimal models. Using each chosen feature with its value in u_1 gives the four prime implicants that are true on u_1:

$$c_1 = \neg\mathsf{clear}_{g,t} \sqcap \mathsf{clear} \sqcap \mathsf{on\text{-}table}, \qquad |c_1| = 5,$$
$$c_2 = \neg\mathsf{clear}_{g,t} \sqcap \mathsf{on\text{-}table} \sqcap \neg\exists \mathsf{on}^{-}.\top, \qquad |c_2| = 7,$$
$$c_3 = \neg\mathsf{clear}_{g,t} \sqcap \mathsf{clear} \sqcap \neg\exists \mathsf{on}.\top, \qquad |c_3| = 7,$$
$$c_4 = \neg\mathsf{clear}_{g,t} \sqcap \neg\mathsf{holding} \sqcap \neg\exists \mathsf{on}.\top \sqcap \neg\exists \mathsf{on}^{-}.\top, \qquad |c_4| = 11.$$

Cheapest cover. The MaxSAT problem of Equations 6–7 is

$$\text{hard:} \quad \big(y_{c_1} \vee y_{c_2} \vee y_{c_3} \vee y_{c_4}\big)$$
$$\text{soft:} \quad \big\{(\neg y_{c_1},\ 5),\ (\neg y_{c_2},\ 7),\ (\neg y_{c_3},\ 7),\ (\neg y_{c_4},\ 11)\big\}$$

with optimum 5, which gives $C_{x_1} \equiv \mathsf{clear} \sqcap \neg\mathsf{clear}_{g,t} \sqcap \mathsf{on\text{-}table}$: "a clear block on the table that is not the goal block". Since $x_1^{(1)}$ has a single positive feature vector, the MaxSAT problem has only one hard clause and is trivial. In general, a vector has several positive feature vectors, which can share prime implicants, and the MaxSAT step is needed to find a cheapest cover.

C EXPERIMENTAL DETAILS AND FULL RESULTS

This appendix gives the benchmark splits, the full decoding and coverage results that Table 1 summarizes, the minimality certificates, the results on unseen embedding vectors and feature vectors, the comparison with per-dimension decoding, and the L1 regularization sweep.

C.1 DATASET STATISTICS

Table 3 lists the number of instances and the range of object counts in the training, validation and test splits of each domain.

Table 3: The benchmark splits. For each domain and split: the number of instances (#I) and the range of object counts (#O). Instances whose goal already holds in the initial state are excluded, as everywhere in the pipeline, so the test counts are the n of Table 5. Training instances are small enough for exhaustive state-space expansion; test instances run to two orders of magnitude more objects.

	train		validation		test	
domain	#I	#O	#I	#O	#I	#O
`blocks-clear`	17	2–7	6	8–9	133	6–477
`blocks-on`	19	2–7	6	7–9	150	5–488
`blocks-on-table`	16	2–7	6	7–9	136	5–488
`childsnack`	19	6–20	5	20–21	164	20–1326
`depots`	264	14–20	84	20–25	279	16–60
`driverlog`	70	7–15	19	15–18	112	14–50
`ferry`	26	3–10	7	10–12	156	7–1461
`gripper`	9	5–13	3	14–16	117	17–133
`miconic`	68	3–21	17	21–27	104	5–681
`satellite`	8	6–12	4	12–12	144	11–402
`spanner`	71	6–24	18	24–28	90	9–833
`transport`	14	6–15	4	15–16	171	12–354

C.2 FULL DECODING AND COVERAGE RESULTS

Table 4 gives the full decoding results, including the consistency of each decoder with the network, and Table 5 gives the full coverage results, including plan quality. Plan quality is mixed: the symbolic policy finds shorter plans than the network on `blocks-clear`, `childsnack`, and `transport`, and longer ones on `blocks-on`, `blocks-on-table`, `depots`, and `driverlog`. The coverage of the ALCSAT variants drops most on `miconic` and `spanner`, where layered ALCSAT has its lowest consistency with the network (0.813 and 0.888).

Table 4: The four decoding methods. *Train consistency* is the fraction of sampled (state, object) pairs on which the decoded concepts reproduce the R-GNN's embeddings. It is *truncated* to three decimals, so 1.000 means *exactly* 1. Concepts that collapsed to $\perp$, decodes that returned nothing, are excluded from the size means.

	mean concept size				train consistency			
domain	ours	IDT	lay.	flat	ours	IDT	lay.	flat
`blocks-clear`	6.2	17.8	7.6	1.0	1.000	1.000	0.990	0.879
`blocks-on`	7.7	27.8	7.5	1.0	1.000	1.000	0.961	0.913
`blocks-on-table`	4.6	12.4	5.5	3.0	1.000	1.000	0.992	0.883
`childsnack`	7.3	55.7	3.7	5.6	1.000	1.000	0.998	0.993
`depots`	22.1	135.1	5.2	5.5	1.000	1.000	0.978	0.943
`driverlog`	23.6	67.1	3.1	3.5	1.000	1.000	0.965	0.904
`ferry`	6.1	13.3	5.5	4.1	1.000	1.000	1.000	0.960
`gripper`	4.2	16.0	3.0	4.1	1.000	1.000	1.000	0.962
`miconic`	9.3	20.5	2.0	1.0	1.000	1.000	0.813	0.968
`satellite`	3.6	5.2	1.9	1.8	1.000	1.000	1.000	1.000
`spanner`	8.5	18.0	3.0	2.5	1.000	1.000	0.888	0.866
`transport`	15.3	46.3	3.9	3.5	1.000	1.000	0.953	0.884
all concepts (mean)	13.1	57.7	**4.4**	4.6	**1.000**	**1.000**	0.965	0.935

Table 5: Test instances solved (*cov*) and plan quality (q) by the R-GNN policy and by the symbolic policy, per decoding method. "n" is the number of test instances in the domain. q is the mean ratio of the policy's plan length to the optimal plan length over the solved instances; it is comparable across setups only on domains where they solve equally many instances. Bold marks the best coverage per domain.

		R-GNN		ours		IDT		lay.		flat	
domain	n	cov	q	cov	q	cov	q	cov	q	cov	q
`blocks-clear`	133	117	1.74	**133**	1.23	**133**	1.23	**133**	1.23	**133**	5.89
`blocks-on`	150	**150**	1.12	149	2.23	149	2.23	**150**	1.92	48	33.05
`blocks-on-table`	136	**136**	1.11	**136**	4.10	**136**	4.10	**136**	4.10	135	2.70
`childsnack`	164	124	1.66	124	1.00	104	1.00	**129**	1.00	124	2.15
`depots`	279	275	1.88	**279**	2.68	278	2.69	255	9.82	254	8.34
`driverlog`	112	**80**	11.04	75	23.22	79	25.35	76	15.57	69	7.49
`ferry`	156	**156**	1.00	**156**	1.00	**156**	1.00	**156**	1.00	**156**	1.00
`gripper`	117	**117**	1.00	**117**	1.00	**117**	1.00	**117**	1.00	**117**	1.00
`miconic`	104	**104**	1.00	**104**	1.00	**104**	1.00	11	50.42	29	62.96
`satellite`	144	**144**	1.00	**144**	1.00	**144**	1.00	**144**	1.00	**144**	1.00
`spanner`	90	**90**	1.00	**90**	1.00	**90**	1.00	0	—	0	—
`transport`	171	150	5.72	**155**	3.89	152	3.89	**155**	5.08	120	19.23
all 12 domains	1756	1643		**1662**		1642		1462		1329	
coverage		93.6%		**94.6%**		93.5%		83.3%		75.7%	

C.3 MINIMALITY CERTIFICATES PER DOMAIN

Table 6 lists, per domain and layer, how many decoded concepts are certified to be minimal, that is, for how many the enumeration of prime implicants finished within the cap of 20,000 (Section 4.2). All first-layer concepts are certified. The uncertified concepts all belong to the second layer of four domains: `childsnack` (7 of 44 certified), `depots` (13 of 55), `transport` (15 of 35), and `driverlog` (43 of 46). In the other eight domains, every concept is certified to be minimal.

Table 6: Certified concepts in the per-vector decoding of Table 4: number of decoded concepts whose enumeration of prime implicants finished within the cap of 20,000, out of all decoded concepts, per layer and in total.

	layer 1		layer 2		total	
domain	certified	concepts	certified	concepts	certified	concepts
`blocks-clear`	6	6	12	12	18	18
`blocks-on`	11	11	16	16	27	27
`blocks-on-table`	9	9	15	15	24	24
`childsnack`	26	26	7	44	33	70
`depots`	18	18	13	55	31	73
`driverlog`	9	9	43	46	52	55
`ferry`	9	9	8	8	17	17
`gripper`	11	11	9	9	20	20
`miconic`	4	4	7	7	11	11
`satellite`	2	2	3	3	5	5
`spanner`	6	6	9	9	15	15
`transport`	13	13	15	35	28	48
all 12 domains	124	124	157	259	281	383

C.4 UNSEEN EMBEDDING VECTORS AND FEATURE VECTORS AT TEST TIME

Table 7 shows how well the concepts generalize to the larger test instances. No test object receives an embedding vector outside the decoding sample, so every test object gets a vector for which a concept exists. This does not mean that the concepts see only familiar inputs: a concept is fitted to the feature vectors of the decoding sample, and a larger instance can produce a feature vector that never occurs there, even when the R-GNN maps it to a known embedding vector. Such feature vectors occur in five domains, all at the second layer. They explain every inconsistency with the R-GNN: on objects whose feature vectors occur in the decoding sample, the concepts are consistent with the R-GNN in all twelve domains. An unseen feature vector does not always lead to an inconsistency; in `miconic` and `transport`, the concepts are correct on all of them.

Table 7: Unseen embedding vectors and unseen feature vectors at test time. n is the number of test instances; *states* the number of states on which the concepts were checked; *objects* the number of (state, object) pairs checked. *vocab* is the number of distinct embedding vectors in the decoding sample; *test* the number of distinct vectors realized on the test objects; *unseen vectors* the distinct test vectors outside the vocabulary and the test objects that realize one. *Unseen features* counts the distinct feature vectors of the test objects that never occur in the decoding sample; *misses* is the number of test objects on which the decoded concepts are inconsistent with the R-GNN. *consistency* is the fraction of objects whose decoded vector equals the R-GNN's, truncated to three decimals so that 1.000 means exactly 1. Columns marked (k) are in thousands. For `childsnack`, the 30 largest test instances were left out due to computational restrictions, hence $n = 134$ here instead of 164.

				vectors		unseen vectors		unseen features		
domain	n	states (k)	objects (k)	vocab	test	distinct	misses (k)	distinct	misses (k)	consistency
`blocks-clear`	133	132	38 694	12	12	0	0	0	0	1.000
`blocks-on`	150	419	124 195	16	16	0	0	0	0	1.000
`blocks-on-table`	136	273	85 204	15	15	0	0	0	0	1.000
`childsnack`	134	14 288	2 126 392	44	40	0	0	40	28 297	0.986
`depots`	279	97	3 745	55	55	0	0	424	6	0.998
`driverlog`	112	141	5 195	46	44	0	0	66	13	0.997
`ferry`	156	22	18 297	8	8	0	0	0	0	1.000
`gripper`	117	25	2 296	9	9	0	0	0	0	1.000
`miconic`	104	13	4 120	7	6	0	0	105	0	1.000
`satellite`	144	188	52 259	3	2	0	0	0	0	1.000
`spanner`	90	20	9 652	9	9	0	0	0	0	1.000
`transport`	171	2 106	489 945	35	34	0	0	32	0	1.000

C.5 Per-vector against per-dimension decoding

In Table 8 we compare the concept sizes for the per-vector decoding that is used throughout the main text of the paper, with a variant that decodes one concept per embedding dimension. On average, decoding per dimension roughly doubles the size of our concepts (25.4 against 13.1) and almost triples that of IDT (163.6 against 57.7). The increase comes mostly from `childsnack`, `depots`, `driverlog`, and `transport`; in `ferry`, `miconic`, and `satellite`, the per-dimension concepts are smaller.

Table 8: Decoding one concept *per embedding vector* (Table 4) compared to decoding one concept *per embedding dimension*: mean concept size. Both regimes reach 1.000 train consistency on every domain. †: the enumeration of prime implicants hit the cap of 20,000 for some dimensions, so our per-dimension sizes are upper bounds.

	per-vector		per-dimension	
domain	ours	IDT	ours	IDT
`blocks-clear`	6.2	17.8	7.0	15.2
`blocks-on`	7.7	27.8	14.5	49.5
`blocks-on-table`	4.6	12.4	9.1	28.3
`childsnack`	7.3	55.7	41.0	195.2
`depots`†	22.1	135.1	85.9	1109.5
`driverlog`†	23.6	67.1	80.9	305.2
`ferry`	6.1	13.3	5.2	9.1
`gripper`	4.2	16.0	6.7	7.4
`miconic`	9.3	20.5	4.9	9.1
`satellite`	3.6	5.2	0.6	1.1
`spanner`	8.5	18.0	8.4	22.0
`transport`†	15.3	46.3	41.0	211.5
all concepts (mean)	**13.1**	57.7	25.4	163.6
train consistency (mean)	**1.000**	**1.000**	**1.000**	**1.000**
decode (CPU h)	4.2	12.7	**1.4**	37.5

C.6 L1 REGULARIZATION

Table 9 compares networks trained with $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}\}$, with the same decoder and policy settings otherwise. Stronger regularization gives fewer distinct embedding vectors (459, 383, and 89 over all domains) and smaller decoded concepts (total size 6,071, 5,027, and 581). At $\lambda = 10^{-3}$, the embeddings collapse to a single vector in seven domains, so the concepts are $\top$ and no policy can be built; in the other five domains, a policy is still obtained. Coverage is highest at $\lambda = 10^{-5}$ (1,704 of 1,756 instances), slightly lower at the $\lambda = 10^{-4}$ used in the main text (1,662), and much lower at $\lambda = 10^{-3}$ (754).

Table 9: L1 with different λ. For each $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}\}$: test instances solved by the symbolic policy (*cov*), plan quality q, number of distinct embedding vectors over both layers (*vectors*), and total complexity of the decoded concepts (*size*). All arms use the same decoder and policy settings as Table 5. †: at $\lambda = 10^{-3}$ the embedding collapsed to a single constant vector in both layers, so the decoded concepts are $\top$ and no policy could be built.

		cov			q			vectors			size		
domain	n	10^{-5}	10^{-4}	10^{-3}	10^{-5}	10^{-4}	10^{-3}	10^{-5}	10^{-4}	10^{-3}	10^{-5}	10^{-4}	10^{-3}
`blocks-clear`	133	133	133	—†	1.22	1.23	—	20	18	2	116	112	2
`blocks-on`	150	150	149	—†	1.35	2.23	—	36	27	2	342	209	2
`blocks-on-table`	136	136	136	—†	3.34	4.10	—	20	24	2	122	110	2
`childsnack`	164	124	124	119	1.00	1.00	5.52	67	70	18	420	513	144
`depots`	279	279	279	267	1.76	2.68	10.98	80	73	19	1230	1613	149
`driverlog`	112	110	75	68	13.32	23.22	19.08	99	55	13	2175	1300	130
`ferry`	156	156	156	156	1.00	1.00	1.00	15	17	15	133	104	110
`gripper`	117	117	117	—†	1.00	1.00	—	16	20	2	114	85	2
`miconic`	104	104	104	—†	1.00	1.00	—	21	11	2	185	102	2
`satellite`	144	144	144	144	1.00	1.00	1.00	11	5	10	68	18	34
`spanner`	90	90	90	—†	1.00	1.00	—	31	15	2	361	128	2
`transport`	171	161	155	—†	6.44	3.89	—	43	48	2	805	733	2
all 12 domains	1756	1704	1662	754				459	383	89	6071	5027	581

D THE SYMBOLIC POLICY FOR GRIPPER

Table 10 gives the definitions of the concepts that the `gripper` policy of Section 6.3 tests, together with the layer-1 concepts they build on. Figure 1 shows the decision tree from which its rules are read, and Table 11 lists the rules as root-to-leaf paths, with the number of training transitions that reach each leaf. Appendix D.1 proves that the policy is optimal on every instance of $\mathcal{Q}_{grip}$ (Theorem 4).

Table 10: Concepts that the policy reads for `gripper`. The decision tree tests the change in the number of objects that satisfy a layer-2 concept, so its entire feature language is the four layer-2 concepts in the lower block, shown below the five layer-1 concepts their definitions name. The other five layer-2 concepts never enter the policy.

concept	definition	reading
$C_1^{(1)}$	$\neg\mathsf{at\text{-}robby} \sqcap \exists \mathsf{at}^-_{g,f}.\top \sqcap \neg\exists \mathsf{at}^-.\top$	the goal room without the robot, goal ball undelivered, with no ball lying in it
$C_2^{(1)}$	$\neg\mathsf{at\text{-}robby} \sqcap \exists \mathsf{at}^-.\top \sqcap \exists \mathsf{at}^-_{g,f}.\top$	the goal room without the robot, goal ball undelivered, with some ball lying in it
$C_5^{(1)}$	free	a free gripper
$C_8^{(1)}$	$\mathsf{at\text{-}robby} \sqcap \neg\exists \mathsf{at}^-_{g,t}.\top \sqcap \neg\exists \mathsf{at}^-_{g,f}.\top$	the robot's room, when it is not the goal room
$C_{10}^{(1)}$	$\exists \mathsf{at}_{g,f}.\top$	the goal ball, undelivered
$C_1^{(2)}$	$C_2^{(1)} \sqcup C_1^{(1)}$	the goal room, while the goal ball is undelivered and the robot is elsewhere
$C_3^{(2)}$	$C_5^{(1)}$	a free gripper
$C_6^{(2)}$	$C_8^{(1)} \sqcap \neg\exists \mathsf{at}^-.C_{10}^{(1)}$	the robot's room, when it is not the goal room and the undelivered goal ball does not lie in it
$C_8^{(2)}$	$C_{10}^{(1)} \sqcup \exists \mathsf{at}^-.C_{10}^{(1)}$	the goal ball, while undelivered, and the room it lies in

Table 11: The symbolic policy for `gripper`. The rules test the changes Δ_i in the number of objects that satisfy the layer-2 concepts $C_i^{(2)}$ of Section 6.3; Table 10 in Appendix D gives their definitions. A rule is the conjunction of the sign tests on its root-to-leaf path in the decision tree (Figure 1). A *fallback* rule applies only when no primary rule matches. *moves matched* names every action the rule accepts. *opt. / non-opt.* gives the optimal and non-optimal training transitions that reached the leaf.

kind	rule	tests	moves matched	opt. / non-opt.
primary	r_1	$\Delta_8 < 0$	pick up the goal ball, or drop it in the goal room	274 / 0
primary	r_2	$\Delta_8 \geq 0, \Delta_1 > 0, \Delta_6 \leq 0$	leave the goal room without the goal ball	180 / 0
primary	r_3	$\Delta_8 \geq 0, \Delta_1 \leq 0, \Delta_6 < 0, \Delta_3 \leq 0$	carry the goal ball into the goal room	124 / 0
fallback	f	$\Delta_8 \geq 0, \Delta_1 \leq 0, \Delta_6 \geq 0, \Delta_3 > 0$	drop a non-goal ball (optimal exactly when the other gripper also holds a non-goal ball)	110 / 540

D.1 PROOF OF THEOREM 4

Theorem 4 (Gripper optimality, proof in Appendix D.1)**.** *On every instance of $\mathcal{Q}_{grip}$, the symbolic policy of Section 6.3 (rules r_1–r_3 and f, Table 11) over the concepts of Table 10 reaches the goal in exactly $V^*(s_0)$ steps.*

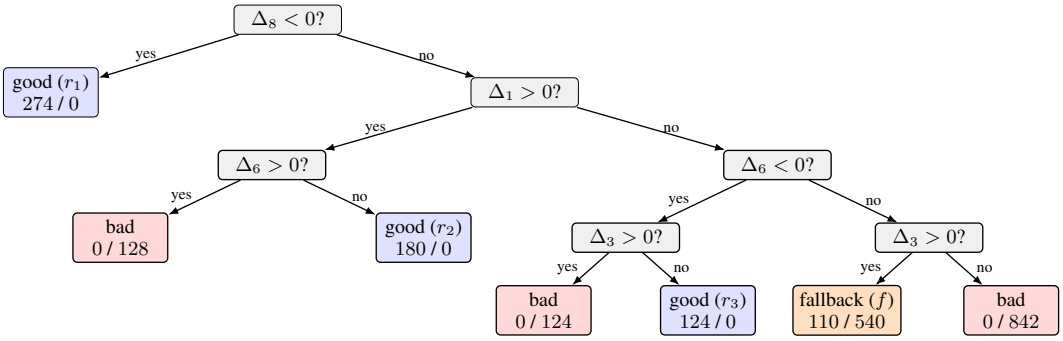


Figure 1: The effect-pattern decision tree learned for `gripper`. Each internal node tests the sign of Δ_i, the change in the number of objects that satisfy the layer-2 concept $C_i^{(2)}$ of Table 10; each leaf gives the number of optimal / non-optimal training transitions that reach it, as labeled by the R-GNN. The root-to-leaf paths are the rules of Table 11.

Table 12: All actions of $\mathcal{Q}_{grip}$ in a non-goal state, the sign (-1, 0 or 1) of the change they cause in each count the tree reads, the rule of Table 11 they fire, and whether they are optimal. Here b^* is the goal ball, r^* the goal room, and r the other room. A dash means that no rule fires and the policy rejects the transition.

action	state	sgn Δ_8	sgn Δ_1	sgn Δ_6	sgn Δ_3	rule	optimal
move $r^* \to r$	b^* lies in r	0	1	0	0	r_2	yes
move $r^* \to r$	b^* carried	0	1	1	0	–	no
move $r \to r^*$	b^* lies in r	0	-1	0	0	–	no
move $r \to r^*$	b^* carried	0	-1	-1	0	r_3	yes
pick b^*		-1	0	1	-1	r_1	yes
pick ordinary ball		0	0	0	-1	–	no
drop b^* in r^*		-1	0	0	1	r_1	yes
drop b^* in r		1	0	-1	1	–	no
drop ordinary ball		0	0	0	1	f	iff the other gripper holds an ordinary ball

Proof. By Table 10, in a non-goal state $|[\![C_8^{(2)}]\!]_s|$ is 2 if b^* lies in a room and 1 if it is carried, $|[\![C_1^{(2)}]\!]_s| = 1$ iff the robot is not in r^*, $|[\![C_6^{(2)}]\!]_s| = 1$ iff the robot is not in r^* and carries b^*, and $|[\![C_3^{(2)}]\!]_s|$ is the number of free grippers; once b^* is dropped in r^*, the first three counts are 0. So both the effect pattern of an action and $V^*(s)$ depend only on the position of b^*, the room of the robot, and the number of free grippers, and not on the number of ordinary balls. Table 12 lists every applicable action with its pattern, the rule it fires, and whether $V^*(s') = V^*(s) - 1$: every action fired by a primary rule is optimal, and every non-optimal action except dropping an ordinary ball fires no rule. The fallback rule f is used only when no primary rule matches any successor, which happens only when the robot is in r with b^* lying in r and both grippers holding ordinary balls; there dropping either ball is optimal. Hence every step reduces V^* by one, and the policy reaches the goal in exactly $V^*(s_0)$ steps. □