Formal Explanations of Neural Network Policies for Planning

This paper is under review for IJCAI. Due to the space limit of the HAXP submission,
we are unable to take into account the comments of the IJCAI reviewers.

Abstract

Deep learning is increasingly used to learn policies for planning problems. However, policies represented by neural networks are difficult to interpret, verify and trust. Existing formal approaches to post-hoc explanations provide concise reasons for a *single* decision made by an ML model. However, understanding planning policies requires explaining *sequences* of decisions. In this paper, we formulate the problem of finding explanations for the sequence of decisions recommended by a learnt policy in a given state. We show that, under certain assumptions, a minimal explanation for a sequence can be computed by solving a number of single decision explanation problems which is linear in the length of the sequence. We present experimental results of our implementation of this approach for ASNets policies for classical planning domains.

1 Motivation

Deep learning has become the method of choice in the areas of AI that focus on perception, and is rapidly gaining traction in other areas that have traditionally been strongholds of reasoning, search, and combinatorial optimisation. In automated planning for instance, new work has emerged that uses deep learning to learn policies and heuristics in a wide range of planning domains. We refer the reader to (Toyer et al. 2018; Groshev et al. 2018; Garg, Bajpai, and Mausam 2020; Zhang and Geißer 2022; Karia and Srivastava 2022) for examples of work aiming at learning policies for planning domains, and to (Shen, Trevizan, and Thiébaux 2020; Ferber, Helmert, and Hoffmann 2020; Karia and Srivastava 2021; Ferber et al. 2022; Gehring et al. 2022) as representatives of work on learning heuristics to guide the search for a plan.

As the use of deep learning becomes more widespread in planning, the need to understand the solutions it produces becomes more pressing. Policies represented by neural networks are notoriously opaque, and difficult to understand, verify, and trust (Toyer et al. 2020; Vinzent, Steinmetz, and Hoffmann 2022). At the minimum, one would like to be able to explain why a particular course of action was recommended by the policy – by identifying the properties of the state of the world prior to the execution of the policy which led to that recommendation – so as to help the user decide whether this recommendation should be trusted. This is the problem addressed in this paper.

1.1 Running Example

Throughout the paper, we use the example of Yvette who needs to take a turnpike to get to her final destination where she will spend a couple of weeks holidays. The turnpike requires to either purchase a weekly pass online or pay with cash at a toll gate. The pass is expensive and should not be taken unless one expects to use the turnpike multiple times in a single week. In our example, we assume that Yvette does not currently have a pass or cash. Hence, the policy prescribes to drive to an ATM, withdraw some money, drive to the toll gate, pay the toll, and drive to the destination.

Why choose this course of action? Firstly because i) Yvette is on the side of the turnpike opposite to her destination. Indeed this course of action would work in any state in which condition i) holds. However, this fails to explain *why* Yvette should not directly go to the toll gate. This is because ii) she has no pass and iii) she has no money. A correct explanation should therefore include all three conditions.

If, instead, Yvette did hold a pass, then the policy would skip the visit to the ATM. The explanation would then mention the fact that Yvette holds a pass but would not mention her lack of money as the policy would still have prescribed this course of action even if she had money.

Explanations can also expose unexpected reasons for decisions. Policies learnt using techniques such as deep learning are not guaranteed to be rational or even valid. They can also expose preferences or bias. For instance, the explanation could include the proposition iv) it is sunny, which implicitly means that the policy would have decided differently if it was not. It is questionable whether iv) is relevant in this context (maybe the idea is that you would not want to make a trip to the ATM under the rain). The neural network could also be prejudiced against certain groups of people and give different advice depending on gender, race, etc. (Darwiche and Hirth 2020).

1.2 Existing Work

The nascent work on explainable planning has been focusing on a different set of problems in a different setting, in particular on model reconciliation and contrastive explanations for conventional model-based planning (Chakraborti, Sreedharan, and Kambhampati 2020). In model reconciliation, the aim is to generate explanations allowing a human user to update his model of the planning problem to make it consistent with the plan produced by a planning agent (Chakraborti et al. 2017; Sreedharan, Chakraborti, and Kambhampati 2021; Vasileiou et al. 2022). This latter question is concerned with the properties of the planning model, rather than those of the policy. The second prominent line of research in the explainable planning literature is the generation of contrastive explanations outlining why the planner chose a course of action over others within the space of possible plans (Eifler et al. 2020; Kasenberg, Thielstrom, and Scheutz 2020; Krarup et al. 2021). These works are concerned with understanding the space of possible decisions and their respective merits, rather than a particular policy.

Therefore, as a starting point, we instead turn to prior work concerned with explaining deep learning and other data-driven models for *classification* tasks. Existing approaches typically either compute simpler models that locally approximate the classifier's behavior (Ribeiro, Singh, and Guestrin 2016; Lundberg and Lee 2017), or identify sufficient conditions on the inputs that led the neural network to produce a particular output (Ribeiro, Singh, and Guestrin 2018; Ignatiev, Narodytska, and Marques-Silva 2019). One approach falling into the latter class is to compute *abductive* explanations that are *minimal* sufficient conditions for the decision. This has the advantage of providing formal guarantees of soundness and non-redundancy (Ignatiev, Narodytska, and Marques-Silva 2019; Darwiche and Hirth 2020; Marques-Silva and Ignatiev 2022).

However, the above approaches are designed to explain a single decision, whereas understanding the recommendations of a planning policy requires explaining why a particular *sequence* of decisions was made. The latter is more challenging as it involves reasoning about repeated applications of the policy network and about the successive changes they induce in the state of the world in which the policy is executed.

1.3 Contribution

In this paper, we extend abductive explanations from single to sequential decisions. We restrict ourselves to classical planning policies and explanations of why the policy makes a certain sequence of decisions from a given state. We formally define explanations for a sequence of decisions, and show that, under certain assumptions, the problem of finding an explanation for the sequence can be decomposed into that of finding explanations for the individual decisions in the sequence. We provide an algorithm that exploits this decomposition to compute a minimal explanation for a sequence by making a number of consistency tests pertaining to individual decisions that is at most linear in the length of the sequence and in the number of state variables. We then discuss the implementation of our approach to explain policies represented by Action Schema Networks (ASNets) (Toyer et al. 2020) and report on its performance on sparse ASNets policies for classical planning domains. We conclude by discussing the limits of our work and possible extensions.

2 Background

We start by introducing the type of planning problems we consider, their representation, and our notations.

Many of the recent work on deep learning for planning assume that the model of the planning domain is available to the learner. We assume that it is also available to the explainer. Here we represent the classical planning instance $I = \langle X, A, g\rangle$ under consideration using the SAS$^+$ formalism (Bäckström and Nebel 1995). X is a set of finite-domain *state variables*, where D_x is the *domain* of variable x. A *partial state* (or partial valuation) s is an assignment of value to a subset $X_s \subseteq X$ of the variables such that $s[x] \in D_x$ for $x \in X_s$. If $X_s = X$ then we say that s is a *state*; we write S for the set of states. A *proposition* $(x = v)$ is a partial valuation assigning a value to a single variable.

The goal g is a partial state. Given two partial states s and s', we write $s \subseteq s'$ when $s[x] = s'[x]$ for all $x \in X_s$. A *completion* of a partial state s is a state s' such that $s \subseteq s'$. The result of applying a partial valuation e to a partial state s is the partial state $s \oplus e$ over $X_s \cup X_e$ defined by $(s \oplus e)[x] = e[x]$ if $x \in X_e$ and $(s \oplus e)[x] = s[x]$ if $x \in X_s \backslash X_e$. We also define the binary operator $\ominus$ over partial states: $s \ominus e$ is the restriction of s to the variables $X_s \backslash X_e$. For a variable $x \in X_s$ will write $s - x$ as an abbreviation for $s \ominus (x = s[x])$.

A is the set of *actions*. Action $a \in A$ is characterised by two partial valuations representing its *precondition* $\text{pre}(a)$ and its *effect* $\text{eff}(a)$, respectively. We say that the action is *applicable* in a state $s \in S$ iff $\text{pre}(a) \subseteq s$ and write $A(s) \subseteq A$ for the subset of actions applicable in s. Moreover, given a partial state s and an action a, the *progression* of s through a is the partial state $\text{prg}_a(s) = s \oplus \text{eff}(a)$. Note that if s is a state and $a \in A(s)$ then $\text{prg}_a(s)$ is the state resulting from applying a in s.

A *policy* for the planning instance is a function $\pi : S \mapsto A$ mapping states to applicable actions, i.e. $\pi(s) \in A(s)$. We define the *n-long trajectory* $\tau^n_\pi(s)$ *induced* by π from state s as follows: $\tau^n_\pi(s) = s_1 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_{n+1}$ such that $s_1 = s$ and for all $0 < i \leqslant n$, $a_i = \pi(s_i)$ and $s_{i+1} = \text{prg}_{a_i}(s_i)$. Finally, the *$n$-long sequence of actions recommended by* π in s is $\pi^n(s) = a_1, \dots, a_n$, where the a_is are the successive actions in $\tau^n_\pi(s)$.

Given a planning instance I, a policy π for I, an *initial state* s, and an integer n, our problem is to explain why π recommended the sequence of actions $\pi^n(s)$ in s. The explanation is meant to shed light on the appropriateness of the recommendation.

The definition of explanations presented in the next section relies on the fact that the policy will recommend the same sequence of actions for certain states. To ensure that for any state s and any length n, a policy can recommend an n-long sequence from s, we make the following assumptions. First, we assume that there is no terminal state, i.e., $A(s) \neq \varnothing$ for all states. This is not a restriction as a default action could be to do nothing. Second, we assume that A includes a special goal action a_g which has no effect and which is only applicable in goal states (this will act as a marker of the goal being reached), and that π is a total function on S such that $\pi(s) = a_g$ iff $g \subseteq s$. This is purely for convenience as policies generally check whether a goal state has been reached before computing the next action. With these assumptions, our theory applies uniformly to any trajectory, regardless of whether it reaches the goal.

3 Explanations of Neural-Network Policies

An explanation of a decision (or sequence of decisions) in a state is a condition on this state that led to this decision being made: the decision was made because the condition was satisfied in this state. Said differently, the same decision would have been made in any other state that satisfies this condition. We could allow arbitrary conditions; e.g. the explanation could be the logical formula that describes exactly all the states in which this decision would be taken. However, such an explanation would not be very helpful. We aim instead for a 'simple' explanation, that is, an explanation that mentions as few propositions as possible and has the simple structure of a conjunction.

We use a definition similar to that of (Marques-Silva and Ignatiev 2022). An explanation is a partial state that entails the decision; in logic, this is known as an implicant.

Definition 1. *An* explanation of a single decision a for policy π *is a partial state* $\mathbf{z}$ *such that* π *yields the same decision for all completions of* $\mathbf{z}$*:*

$$\forall s \in S. \quad (\mathbf{z} \subseteq s) \implies \pi(s) = a.$$

When s completes $\mathbf{z}$, we say that $\mathbf{z}$ *explains* decision a in s.

Our goal is not to explain just the first decision of the policy, but the complete sequence of decisions. While the first decision was based on the initial state, later decisions were made based on the later states. These states, however, are fully determined by the initial state and the actions taken. Using the planning model, it is therefore possible to trace the sufficient condition that led to the full sequence of actions back to the initial state.

Definition 2. *An* explanation of the n-long sequence of decisions $a_1, \ldots, a_n$ for a policy π *is a partial state* $\mathbf{z}$ *such that* π *yields the same sequence of decisions for all completions of* $\mathbf{z}$*:*

$$\forall s \in S. \quad (\mathbf{z} \subseteq s) \implies \pi^n(s) = a_1, \ldots, a_n.$$

Similarly as before, when s completes $\mathbf{z}$, we say that $\mathbf{z}$ *explains* the sequence of decisions $a_1, \ldots, a_n$ in s. We note that if $a_n = a_g$ is the goal action, then the sequence $a_1, \ldots, a_{n-1}$ leads all completions of $\mathbf{z}$ to a goal state since π only recommends applicable actions and a_g is only applicable in a goal state. The sequence of actions recommended by the policy might not lead to the goal; in this case, the loop of the infinite trajectory induced by the policy does not occur at a goal state. If one wants to compute an explanation for this infinite sequence, it is possible to use Definition 2 with $n = L \times |S|$ where L is the length of the loop and $|S|$ the total number of states: if $\mathbf{z}$ explains $\pi^n(s)$, then it explains $\pi^{n+k}(s)$ for all $k \geqslant 0$. It may be possible to derive better bounds.

It should be clear that there can be multiple explanations in the same state. For instance, in our running example, Yvette needs either a pass or some cash to take the turnpike. Both the fact that she has a pass and the fact that she has cash would be acceptable explanations for driving directly to the toll gate. In a state where she has a pass and cash, these two explanations are therefore suitable. We also note that explanations enjoy monotonic properties: if $\mathbf{z}$ is an explanation, any superset of $\mathbf{z}$ is an explanation. In particular, the complete initial state is an explanation, although hardly a useful one.

Our goal is to compute the 'best' explanation for the sequence of decisions made from our initial state. Specifically, we want to compute a subset-minimal explanation (akin to a *prime* implicant in logic), i.e., an explanation $\mathbf{z}$ such that no strict subset of $\mathbf{z}$ is an explanation. Minimal explanations provide additional benefits: all variables mentioned in the explanation are required, in the sense that if any were removed, the partial state would no longer be an explanation. Therefore, seeing a variable that should not be relevant in a minimal explanation should raise questions about the policy, while this phenomenon is unsurprising in a non-minimal explanation.

Definition 3. *Given a policy* π*, an integer* n*, and a state* s*, the* minimal explanation problem *is to find a minimal partial state that explains the sequence of decisions* $\pi^n(s)$ *in* s*.*

Ignatiev, Narodytska, and Marques-Silva (2019) have shown how to compute explanations for single decisions. The policy is translated into a set of constraints C_π over a set of variables that includes the state variables X which are the input to the policy, and the variable y which represents its output. The model of C_π are exactly all the pairs $\langle s, y \rangle$, $s \in S$, $y = \pi(s)$. If π is represented by a neural network, C_π can be formulated as a set of mixed-integer programming or sat modulo theory constraints (see Section 5). In order to decide whether $\mathbf{z}$ is an explanation for a decision a, the constraints $\mathbf{z}$ and $\neg d$ are added to the set where $d \equiv (y = a)$. If the resulting set of constraints is consistent, then there exists a state s' that completes $\mathbf{z}$ and yields a decision different from a; hence, $\mathbf{z}$ does not explain a. Otherwise, all states that complete $\mathbf{z}$ lead to decision a, and $\mathbf{z}$ explains a:

$$\mathbf{z} \text{ explains } a \iff \neg\mathsf{CO}(C_\pi \wedge \mathbf{z} \wedge \neg d).$$

Using monotonicity, it is then possible to greedily search for a minimal explanation. This is done by starting with an existing explanation $\mathbf{z}$, for instance the initial state s, and testing whether for some variable $x \in X_{\mathbf{z}}$, $\mathbf{z}' := \mathbf{z} - x$ remains an explanation. If $\mathbf{z}'$ indeed explains a, we replace $\mathbf{z}$ with it; otherwise we move to the next variable x until we tried to remove each variable. The same variable does not need to be tested more than once.

4 Computing a Minimal Explanation

We now turn to the problem of computing a minimal explanation for a sequence of decisions.

4.1 Naive Algorithm

For completeness, we first consider a naive algorithm, illustrated in Figure 1. We expect that this algorithm will be impractical, as it requires testing the consistency of constraint sets involving too many variables.

Similarly as in the single decision case, the idea of the algorithm is to build a single set of constraints which is consistent iff a specified partial state does not explain a specified sequence of decisions $a_1, \ldots, a_n$. Given an initial state s_1, we define the set of constraints $C_{a_1, \ldots, a_n}$ which computes the states $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} s_n$ reached by applying the successive actions as well as the decisions $y_i = \pi(s_i)$ of the policy in each of these states; we then compare the y_is with

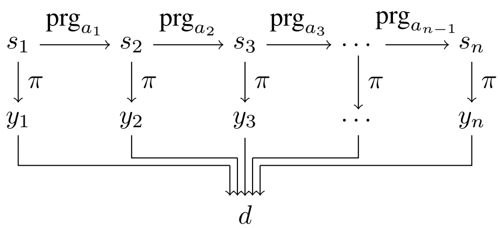

Figure 1: Graphical representation of $C_{a_1,\dots,a_n}$, the set of constraints used to determine whether a partial state is an explanation. Nodes of the graph are sets of variables. Arrows represent constraints defined such that the target variables are a function of the source variables.

the a_i:

$$\begin{array}{ll} s_{i+1} = \mathrm{prg}_{a_i}(s_i) & \forall i \in \{1, \dots n-1\} \\ y_i = \pi(s_i) & \forall i \in \{1, \dots n\} \\ d_i = (y_i = a_i) & \forall i \in \{1, \dots n\} \\ d = \bigwedge_{i=1}^{n} d_i & \end{array}$$

Finally, we add the constraints that s_1 should complete $\mathbf{z}$ and that d should be false. The resulting set of constraints,

$$C_{a_1,\dots,a_n} \wedge \bigwedge_{x \in X_{\mathbf{z}}} (s_1[x] = \mathbf{z}[x]) \wedge \neg d,$$

is consistent iff there exists a state s_1 that completes $\mathbf{z}$ for which π generates a different sequence of decisions than the specified one; i.e., $\mathbf{z}$ does not explain $a_1, \dots, a_n$. As before, one can then use a greedy algorithm to compute the explanation, starting with $\mathbf{z} = s_1$ and progressively removing propositions.

However in this set of constraints, the variables include n duplicates of the set of state variables X, which are needed to represent the successive states $s_1 \dots s_n$, as well as the much larger set of variables required to represent n duplicates of the computation of π (via the constraints C_π mentioned above). This makes testing consistency impractical for anything but very small sequences.

4.2 Forward Decomposition

We now show that it is possible to decompose the explanation so that it is easier to compute a single minimal explanation. This decomposition leads to an algorithm that need only solve a number of single decision explanation problems that is in the worst case linear in the length of the sequence. In the following $\circ$ is the function composition.

Theorem 1. *Let π be a policy, s be a state, and let $\tau_\pi^n(s) = s_1 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_{n+1}$ be the n-long trajectory induced by π from s. Let $\mathbf{z} \subseteq s$ be a partial state and let $\mathbf{z}_1, \dots, \mathbf{z}_n$ be a sequence of partial states defined by $\mathbf{z}_{i+1} = prg_{a_i} \circ \dots \circ prg_{a_1}(\mathbf{z})$. Then $\mathbf{z}$ explains $\pi^n(s)$ iff for all steps i, $\mathbf{z}_i$ explains a_i in s_i.*

The proof of Theorem 1 is in Appendix A. Theorem 1 gives us a clear procedure to verify whether a partial state $\mathbf{z}$ is an explanation, which is to compute the partial states $\mathbf{z}_i$ resulting from applying the actions from $\mathbf{z}$ (note: $\mathbf{z}_{i+1} = \mathrm{prg}_{a_i}(\mathbf{z}_i)$) and to verify whether each $\mathbf{z}_i$ explains a_i.

Algorithm 1: Computing a minimal explanation for a sequence of decisions.

1: **procedure** MINIMALEXPLANATION(I, π, n, s)
2: $\quad s_1 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_{n+1} = \tau_\pi^n(s)$
3: $\quad \mathbf{z}_1 := s$
4: $\quad$ **for** $i \in \{1, \dots, n-1\}$ **do**
5: $\quad\quad \mathbf{z}_{i+1} := \mathrm{prg}_{a_i}(\mathbf{z}_i)$
6: $\quad$ **for** $x \in X$ **do**
7: $\quad\quad explains := True$
8: $\quad\quad$ **for** $i \in \{1, \dots, n\}$ **do** ▷ Testing condition of Th. 1
9: $\quad\quad\quad \mathbf{z}_i := \mathbf{z}_i - x$
10: $\quad\quad\quad$ **if** $\mathsf{CO}(C_\pi \wedge \mathbf{z}_i \wedge \neg d_i)$ **then**
11: $\quad\quad\quad\quad explains := False$
12: $\quad\quad\quad\quad$ **break**
13: $\quad\quad\quad$ **if** $x \in X_{\mathrm{eff}(a_i)}$ **then**
14: $\quad\quad\quad\quad$ **break** ▷ Can stop now (cf. Eq. 1)
15: $\quad\quad$ **if** $\neg explains$ **then** ▷ Must reinsert x
16: $\quad\quad\quad$ **for** $j \in \{1, \dots, i\}$ **do** $\mathbf{z}_j := \mathbf{z}_j \oplus (x = s_j[x])$
17: $\quad$ **return** $\mathbf{z}_1$

In the greedy algorithm that we propose next, we use the additional property: for any variable x, $\mathrm{prg}_{a_i} \circ \dots \circ \mathrm{prg}_{a_1}(\mathbf{z} - x) =$

$$\begin{cases} \mathrm{prg}_{a_i} \circ \dots \circ \mathrm{prg}_{a_1}(\mathbf{z}) & \text{if } x \in X_{\mathrm{eff}(a_i)} \cup \dots \cup X_{\mathrm{eff}(a_1)} \\ \mathrm{prg}_{a_i} \circ \dots \circ \mathrm{prg}_{a_1}(\mathbf{z}) - x & \text{otherwise.} \end{cases} \quad (1)$$

The first case is useful because it means $\mathrm{prg}_{a_i} \circ \dots \circ \mathrm{prg}_{a_1}(\mathbf{z} - x)$ explains a_{i+1} iff $\mathrm{prg}_{a_i} \circ \dots \circ \mathrm{prg}_{a_1}(\mathbf{z})$ does; so as soon as variable x appears in the effects of a_i, we will be able to ignore the condition of Theorem 1 that z_k should explain a_k in s_k for all $k > i$. The second case is useful because it makes it easy to compute $\mathrm{prg}_{a_i} \circ \dots \circ \mathrm{prg}_{a_1}(\mathbf{z} - x)$ from $\mathrm{prg}_{a_i} \circ \dots \circ \mathrm{prg}_{a_1}(\mathbf{z})$.

Our procedure is described in Algorithm 1. In Lines 3–5, the algorithm initialises the variables $\mathbf{z}_i$, mirroring the variables from Theorem 1 for explanation $\mathbf{z} := s$. Line 6 starts the loop in which each variable x is tentatively removed from the explanation. The variable $explains$ will record whether $\mathbf{z} - x$ explains the sequence; until proven otherwise, it is assumed it does. The inner loop from Line 8 verifies the condition of Theorem 1. After $\mathbf{z}_i$ is updated (using the second case of Eq. 1), the condition is verified on Line 10. If $\mathbf{z}_i$ does not explain action a_i, the inner loop is stopped; the loop from Line 16 will then undo the update. Otherwise, the inner loop moves to the next step $i + 1$ *except* if variable x is mentioned in the effects of a_i in which case the loop can be stopped (first case of Eq. 1).

This algorithm requires $O(n \times m)$ calls to the consistency solver where n is the length of the sequence of m is the number of state variables.

We illustrate this algorithm with our running example. We assume the policy described in Fig. 2 is used. The initial state, and so the initial explanation, is $\mathbf{z} = \{$*Friday*, *Sunny*, *NoMoney*, *NoPass*, *AtHome*, $\dots\}$.

Algorithm 1 starts by removing *Friday* from the explanation. The partial state is therefore $\mathbf{z}' = \mathbf{z}'_1 = \mathbf{z} \ominus \{$*Friday*$\}$.

1. If has passed the TollGate, has a pass, or has some money, drive towards the destination.
2. Otherwise, if not at the ATM, drive towards the ATM.
3. Otherwise, if it is sunny, withdraw money.
4. Otherwise, purchase a pass online.

Figure 2: The policy given to Yvette (without goal action a_g)

We find that it is impossible to instantiate the only free variable (day of the week) in such a way that a decision different from *DriveToATM* is taken; in other words, $\mathbf{z}'_1$ explains the first decision. Progressing $\mathbf{z}'_1$ gives us $\mathbf{z}'_2 = \{\mathit{Sunny}, \mathit{NoMoney}, \mathit{NoPass}, \mathit{AtATM}, \dots\}$. This partial state also explains the second decision (*WithdrawMoney*), and so on until the end of the sequence. Therefore, $\mathbf{z}'$ explains the whole sequence.

Then, Algorithm 1 removes *Sunny* from the explanation. The partial state is therefore $\mathbf{z}'' = \mathbf{z}''_1 = \mathbf{z}' \ominus \{\mathit{Sunny}\}$. Partial state $\mathbf{z}''$ explains the first decision (*DriveToATM*). Progressing $\mathbf{z}''_1$ gives us $\mathbf{z}''_2 = \{\mathit{NoMoney}, \mathit{NoPass}, \mathit{AtATM}, \dots\}$. This time, we find that a state in which Yvette is at the ATM and the weather is rainy will yield a different decision (*PurchasePass*) than the second decision (*WithdrawMoney*). Therefore, $\mathbf{z}''$ does not explain the sequence and *Sunny* is added back to the explanation. Similarly, the algorithm would then try and fail to remove the 3 remaining propositions.

5 Implementation

We use Algorithm 1 to explain the recommendations of ASNets policies (Toyer et al. 2018) for classical planning domains. This requires implementing the consistency test in line 10 of the algorithm for ASNets, which have a complex structure and nonlinear activation functions. Our encoding, presented below, is supported by mixed integer programming (MIP) solvers such as Gurobi (Gurobi Optimization, LLC 2023).

5.1 ASNets

ASNets are neural networks dedicated to planning, trained to produce policies that apply to any problem instance from a given planning domain modelled in (P)PDDL (Younes and Littman 2004). An ASNet consists of L alternating *action* and *proposition layers*, beginning and ending with an action layer. In each action layer (resp. proposition layer), each action (resp. proposition) of the planning instance is represented by an action *module* (resp. proposition module). Modules in one layer are connected to *related* modules in the next layer, where a proposition p and action a are related, written $R(a, p)$ iff p appears in the precondition or effect of a. This enables relevant information to be efficiently propagated through the network.

ASNets is capable of learning policies that generalise to problem instances of different size from the same domain thanks to a weight sharing mechanism whereby the action (resp. proposition) modules from the same layer that are ground instances of the same action schema (resp. the same predicate), have the same weights. Here we omit details such as the use of skip connections, and the more complex definition of relatedness in (Toyer et al. 2020), but our implementation supports them.

Action Layers At layer l, excluding the 1st and last layer, the module for action a takes as input a vector u_a^l which is constructed by enumerating the propositions $p_1, \dots, p_M$ that are related to a and concatenating the outputs $\psi_{p_1}^{l-1}, \dots, \psi_{p_M}^{l-1}$ of these propositions' modules from the previous layer. That is $u_a^l = [\psi_{p_1}^{l-1} \dots \psi_{p_M}^{l-1}]^T$. The output of the module is the vector $\phi_a^l = f(W_a^l u_a^l + b_a^l)$ where W_a^l is a weight matrix, b_a^l a bias vector, and f a nonlinearity. ASNets uses exponential linear units (ELU), i.e. $f(x) = x$ if $x \geqslant 0$ and $e^x - 1$ otherwise. Note that except for the final layer, the output vectors of all modules in the network have the same dimension d.

Proposition Layers At layer l, the module for proposition p takes as input a vector v_p^l constructed by enumerating the actions that are related to the proposition, pooling from the outputs of their modules at layer l if they share the same action schema, and concatenating the results. That is $v_p^l = [\mathrm{pool}(\{\phi_a^l \mid \mathrm{op}(a) = o_1 \wedge R(a,p)\}) \dots \mathrm{pool}(\{\phi_a^l \mid \mathrm{op}(a) = o_S \wedge R(a,p)\})]^T$, where pool represents max-pooling and $\mathrm{op}(a)$ is the action schema of action a. Similarly as for action modules, the output of the proposition module is the vector $\psi_p^l = f(W_p^l v_p^l + b_p^l)$.

First Layer The input to an action module of the first layer are the boolean values (0 or 1) of all its related propositions in the current state, booleans indicating whether each of these propositions is true in the goal, and a boolean indicating whether the action is applicable. That is the input vector is $u_a^1 = [\sigma\ \gamma\ \epsilon]^T$ where for all propositions $p_1, \dots, p_M$ related to the action, $\sigma \in \{0,1\}^M$ with $\sigma_j = 1$ iff $p_j \in s$, $\gamma \in \{0,1\}^M$ with $\gamma_j = 1$ iff $p_j \in g$, and $\epsilon = 1$ iff $a \in A(s)$.

Last Layer The output of an action module in the final layer $l = L$ is a single logit ϕ_a^L representing the unnormalised probability that this action should be taken in the current state s given as input to the network. When, as in this paper, a deterministic policy is sought, the applicable action $a \in A(s)$ with maximum ϕ_a^L is selected by the policy.

5.2 MIP Encoding

The main purpose of the encoding is to answer consistency queries where only *some* of the inputs to the ASNets are given, and its output is constrained. The key decision variables in the MIP model fall into 3 categories: variables representing the network inputs, its outputs, and the action and proposition modules. As is well known from other MIP encodings of neural networks, one also needs auxiliary variables to encode the activation functions. Parameters include the weight matrices and bias vectors described above. The MIP has no objective function since we are only testing for consistency.

Policy Inputs We encode each element in the input of an ASNet (current state propositions, goal proposition, applicable actions) using the following binary variables: S_p is true iff proposition p is true in the current state, G_p is true iff proposition p is in the goal, and E_a is true iff a is applicable in the current state. Since a key feature of the MIP model is to allow specifying *partial* states as input, we must add constraints capturing when actions are applicable: action a is applicable when $S_p = 1$ for all $p \in \text{pre}(a)$.

$$\begin{array}{ll} E_a \leqslant S_p & \forall a \in A, \forall p \in \text{pre}(a) \\ E_a \geqslant 1 - |\text{pre}(a)| + \sum_{p \in \text{pre}(a)} S_p & \forall a \in A \end{array}$$

Moreover, ASNets only supports *boolean* propositions $p \equiv (x = v)$ for $x \in X$ and $v \in D_x$. To encode SAS^+ state variables, we add mutex constraints ensuring that at most one proposition assigning a value to a given variable can be true.

$$\begin{array}{ll} \sum_{v \in D_x} S_{(x=v)} \leqslant 1 & \forall x \in X \\ \sum_{v \in D_x} G_{(x=v)} \leqslant 1 & \forall x \in X \end{array}$$

Action Modules To encode action modules, we define the MIP variables $(\text{PAC}^l_a)_i$ representing the ith element of the pre-activation vector (omitting the non-linearity f) of the module for action a in layer l, and $(\text{OUT}^l_a)_i$ representing the ith element of the output vector ϕ^l_a of this module. For an action a with related propositions $p_1, \ldots, p_M$, we have, taking $j = (m-1)d + k$

$$(\text{PAC}^l_a)_i = \sum_{m=1}^{M} \sum_{k=1}^{d} (W^l_a)_{i,j} \cdot (\text{OUT}^{l-1}_{p_m})_k + (b^l_a)_i \text{ for } l > 1$$

$$(\text{PAC}^1_a)_i = \sum_{m=1}^{M} (W^1_a)_{i,m} \cdot S_{p_m} + (W^1_a)_{i,M+m} \cdot G_{p_m} + (W^1_a)_{i,2M+1} \cdot E_a + (b^1_a)_i$$

Proposition Modules Similarly, we define $(\text{PAC}^l_p)_i$ and $(\text{OUT}^l_p)_i$ for each proposition p. For proposition modules we additionally need variables $(\text{POOL}^l_{p,o})_i$ to represent the ith element of the pooled vector over actions related to p sharing the action schema o. If related actions belong to S action schemas $o_1, \ldots o_S$, we have, taking $j = (s-1)d + k$

$$\begin{aligned} (\text{PAC}^l_p)_i &= \sum_{s=1}^{S} \sum_{k=1}^{d} (W^l_p)_{i,j} \cdot (\text{POOL}^l_{p,o_s})_k + (b^l_p)_i \\ (\text{POOL}^l_{p,o})_i &= \max(\{(\text{OUT}^l_a)_i \mid \text{op}(a) = o \wedge R(a,p)\}) \end{aligned}$$

Activations The encoding of f as the ELU function is very similar to the encoding of ReLU in MIP (Fischetti and Jo 2018) and is the same for proposition and action modules. Given a proposition or action b we define the auxiliary variables $(t^l_b)_i$ and $(s^l_b)_i$ to store the positive and negative component of the variable $(\text{PAC}^l_b)_i$, respectively, and $(z^l_b)_i$ which is an indicator variable for the sign of $(\text{PAC}^l_b)_i$. This leads to the following constraints (note that Gurobi linearises the exponential, see (Gurobi Optimization, LLC 2023, p584-586))

$$\begin{array}{cc} (z^l_b)_i = 1 \implies (t^l_b)_i \leqslant 0, & (z^l_b)_i = 0 \implies (s^l_b)_i \leqslant 0 \\ (t^l_b)_i \geqslant 0, & (s^l_b)_i \geqslant 0 \\ (\text{PAC}^l_b)_i = (t^l_b)_i - (s^l_b)_i, & (\text{OUT}^l_b)_i = (t^l_b)_i + e^{-(s^l_b)_i} - 1 \end{array}$$

Policy Output The chosen action is the applicable action a maximising the output ϕ^L_a of the final layer's modules. We enforce this using the following additional variables: C_a which is a binary variable true iff action a is chosen, and $Vmax$ which is the maximal value of ϕ^L_a for *applicable* actions.

$$\begin{array}{ll} \sum_{a \in A} C_a = 1 & \forall a \in A \\ C_a = 1 \implies E_a = 1 & \forall a \in A \\ E_a = 1 \implies Vmax \geqslant \text{OUT}^L_a & \forall a \in A \\ C_a = 1 \implies Vmax \leqslant \text{OUT}^L_a & \forall a \in A \end{array}$$

Temporary Constraints The above constraints constitute the encoding C_π of the ASNet policy and only needs to be built once for a given sets X and A of state variables and actions. For each consistency query involving a planning instance $I = \langle X, A, g \rangle$, an action decision a, and a candidate explanation $\mathbf{z}$, it suffices to temporarily add the following constraints to set g as the goal, prevent a to be chosen by the policy, and look for a possible completion s of $\mathbf{z}$.

$$\begin{array}{ll} C_a = 0 & \\ S_{(x=\mathbf{z}[x])} = 1 & \forall x \in X_{\mathbf{z}} \\ G_{(x=g[x])} = 1 & \forall x \in X_g \\ \sum_{v \in D_x} G_{(x=v)} = 0 & \forall x \in X \backslash X_g \end{array}$$

6 Experimental Results

The goal of our experiments is to evaluate how effective abductive explanations are in explaining why a policy recommends a particular course of actions. In particular, we consider what percentage of the input appears in the minimal explanation. The smaller the explanation, the easier it is for a human to interpret. We also focus on the scalability of Algorithm 1 as the size of the policy increases. For reproducibility, the repository [omitted for anonymity] provides our algorithm implementation, benchmarks used, learnt policies, and the scripts to learn them and run the experiments.

6.1 Experimental Setup

Hardware All experiments were run on a machine with an AMD Ryzen Threadripper 3990X CPU, with 64 cores/128 threads, a clock speed of 2.9 GHz base, 4.3 GHz max boost, and 128 GB of memory.

MIP Configuration Gurobi is the MIP solver used for the experiments. To ensure the model is accurate enough for our experiments, we set the integer feasibility tolerance (`IntFeasTol`) to 10^{-9} and the error for function approximations (`FuncPieceError`) to 10^{-6}. Presolve is also turned off, as our use case is assessing the feasibility of the model rather than finding an optimal value. If a single call to MIP exceeded 2 hours, the algorithm was terminated for the particular problem instance.

ASNet policies We took all deterministic domains and training instances from the code distributions of (Toyer et al. 2020) and (Steinmetz et al. 2022). We ran the script provided by Toyer using 1 core, an 8h timeout, and 64gb of memory to learn, from these instances, two-layer (i.e. two proposition layers and three action layers) *sparse* policies with skip connections and without heuristic inputs. As described in (Toyer et al. 2020, Sec. 6), the ℓ_1-regulariser used to train sparse policies results in many modules having coefficient so close to zero that they are insignificant to the output of the network and are prunned by the ASNet sparsification procedure. Hence, these modules do not appear in the MIP model, making a simpler model to solve.

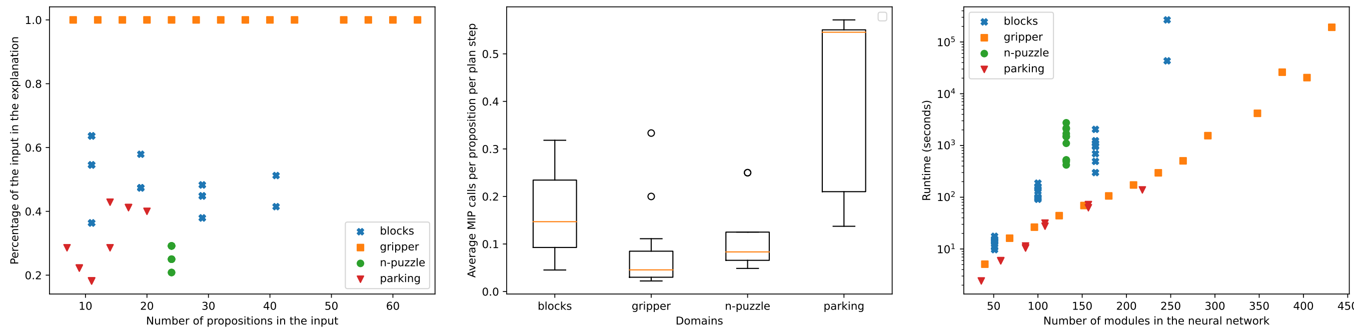

Figure 3: Experimental Results

6.2 Domains and Problems

We kept the domains for which the learnt policy could solve any problem within 150 execution steps, amongst a test set of 20 randomly generated small problems for that domain. This resulted in 4 policies for the domains of Blocksworld, Gripper, n-Puzzle, and Parking, which are present in the ASNet distribution. In order to evaluate our approach, we randomly generated, for each of these domains, the set of problems described below. Explanations were only computed for problems where the policy was able to reach the goal.

Blocksworld Experiments were conducted on 10 problems of each size (2, 3, 4 and 5 blocks) for a total of 40 problems. The policy was able to solve 90% of problems, and out of these, 83% had explanations computed within the time limit.

Gripper Explanations were computed using problems with 2 rooms and 1-15 balls. The policy was able to solve all problems, and explanations could be computed for 93%.

n-Puzzle Experiments were conducted on all solvable and non-trivial combinations of 3-puzzle problems (on a 2x2 grid). Out of these 11 problems, the policy was correctly able to solve all instances and compute all explanations.

Parking Problems in the parking domain comprised of 2 cars and 2-4 curbs. Out of the 22 total problems, the policy reached the goal for 55% and we were able to compute explanations for 92% of these within the time limit.

6.3 Results

Figure 3 shows the size of the explanation as a percentage of the input (left-hand side graph), the average number of calls to the MIP solver per proposition and step in the plan (middle graph), and the CPU time of the algorithm (on a log scale) as a function of the size of the neural network policy for the problem (right-hand side graph).

Size of Explanations In all domains except Gripper, Algorithm 1 produces an explanation smaller than the ASNet's input. For n-Puzzle, its small explanations are due to many static propositions. The grid in the problem is defined as propositions listing the neighbours of each position. As this grid is static, this will not appear in the explanation. On the other hand, no non-trivial explanations were found for Gripper, as the position of the gripper, the location of the balls, and whether or not the gripper is holding anything are all relevant to the plan. Propositions stating which ball the gripper is carrying are not necessary, but they have already been removed from the MIP model due to the sparsity of the network.

Number of Consistency Tests. The average number of calls to the MIP solver per proposition and per action in the plan is always between 0 and 1. It represents how long propositions survive through the inner loop before being reinstated in the explanation or definitely ruled out. When that number is low, the algorithm quickly decides to keep the propositions in the explanation because of satisfiable consistency tests, or quickly decides to remove them because they are achieved by the effect of one of the first few actions. Gripper falls in the former camp: propositions are quickly ruled out as needing to be kept by the consistency test. n-puzzle falls in the latter camp: propositions quickly get removed from the explanation. In Blocksworld, and especially Parking, many propositions are removed from the explanation, but this happens much later in the sequence.

Scalability Much of the runtime is spent in MIP consistency tests. As can be seen from the size of problems we can address, reasoning about neural networks using MIP is a serious bottleneck. Across domains, we have observed that, as input propositions progressively get removed from the explanation, consistency tests generally become harder. This is because proving inconsistency generally becomes harder, whilst consistency becomes easier, but proving consistency leads us to reinsert the proposition, so the algorithm never reaches the easy region of the consistent side of the phase boundary. This behaviour is common with optimisation problems.

There are differences in scalability amongst the domains however. Gripper scaled the best, which is due to each iteration of the inner loop terminating early, as can be seen from the right-hand graph of Figure 3. The many MIP calls to the experiments for Parking problems increased the runtime significantly, resulting in the timeout of larger problems.

7 Conclusion, Limits, and Future Work

We have extended abductive explanations to sequences of decisions recommended by neural network policies. Our decomposition approach to find a single minimal explanation incurs no overahead in comparison with the single decision case, once the length of the sequence is factored in.

Our approach makes a number of limiting assumptions which we discuss here together with possible extensions and future work. The first assumption is the availability of a symbolic planning model. An interesting avenue for future work is the extension of this approach to **learnt planning models**, also represented as neural networks.

We also assumed that actions have simple, unconditional effects. **Conditional effects** can be handled by our naive algorithm. However, Theorem 1 does not allow for them because it is impossible to apply conditional effects to a partial state. We assumed that we have no **background knowledge**, i.e. constraints that restrict the set of possible states (Thiébaux, Hoffmann, and Nebel 2005; Rintanen 2017). These can greatly simplify explanations. Yu et al. (2023) showed that in the single decision case, adding the background knowledge K as another conjunct to the consistency tests performed by the greedy algorithm preserves the minimality of explanations. This property carries over to the multiple decision case and our naive algorithm. However, our decomposition algorithm may not return a minimal explanation with this augmented consistency test, because the set of reachable states cannot be represented by intersections of partial states with K. We leave an efficient treatment of conditional effects and background knowledge for future work.

We have assumed that the actions and the policy are deterministic. Handling **stochastic actions and/or policies** could be achieved by generating explanations that pertain to a finite tree of actions reachable with non-zero probability from the initial state. This would require applying progression and consistency tests at each branch of the tree. Handling stochastic policies would additionally require a more complex consistency test, as the decision d becomes a probability distribution over actions and the test must establish that it is not possible for the policy to return a different distribution.

Another avenue for future work is the generation of all (set-inclusion) **minimal explanations** and of **minimum cardinality explanations**. This is likely to require a different decomposition of the problem than the one presented here, as well as effective heuristics to guide search. Finally, even in the single decision case, methods for computing formal explanations of neural networks suffer from scalability issues due to the expensive consistency test. New breakthroughs in MIP and SMT methods for analysing neural networks, new problem abstractions, and approximate explanation methods will be needed for these approaches to flourish.

References

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Comput. Intell.*, 11: 625–656.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2020. The Emerging Landscape of Explainable Automated Planning & Decision Making. In *Proc. IJCAI*, 4803–4811.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proc. IJCAI*, 156–163.

Darwiche, A.; and Hirth, A. 2020. On the Reasons Behind Decisions. In *Proc. ECAI*, 712–720.

Eifler, R.; Steinmetz, M.; Torralba, Á.; and Hoffmann, J. 2020. Plan-Space Explanation via Plan-Property Dependencies: Faster Algorithms & More Powerful Properties. In *Proc. IJCAI*, 4091–4097.

Ferber, P.; Geißer, F.; Trevizan, F. W.; Helmert, M.; and Hoffmann, J. 2022. Neural Network Heuristic Functions for Classical Planning: Bootstrapping and Comparison to Other Methods. In *Proc. ICAPS*, 583–587.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *Proc. ECAI*, 2346–2353.

Fischetti, M.; and Jo, J. 2018. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3): 296–309.

Garg, S.; Bajpai, A.; and Mausam. 2020. Symbolic Network: Generalized Neural Policies for Relational MDPs. In *Proc. ICML*, 3397–3407.

Gehring, C.; Asai, M.; Chitnis, R.; Silver, T.; Kaelbling, L. P.; Sohrabi, S.; and Katz, M. 2022. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. In *Proc. ICAPS*, 588–596.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning Generalized Reactive Policies Using Deep Neural Networks. In *Proc. ICAPS*, 408–416.

Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/current/refman.pdf.

Ignatiev, A.; Narodytska, N.; and Marques-Silva, J. 2019. Abduction-Based Explanations for Machine Learning Models. In *Proc. AAAI*, 1511–1519.

Karia, R.; and Srivastava, S. 2021. Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning. In *Proc. AAAI*, 8064–8073.

Karia, R.; and Srivastava, S. 2022. Relational Abstractions for Generalized Reinforcement Learning on Symbolic Problems. In *Proc. IJCAI*, 3135–3142.

Kasenberg, D.; Thielstrom, R.; and Scheutz, M. 2020. Generating Explanations for Temporal Logic Planner Decisions. In *Proc. ICAPS*, 449–458.

Krarup, B.; Krivic, S.; Magazzeni, D.; Long, D.; Cashmore, M.; and Smith, D. E. 2021. Contrastive Explanations of Plans through Model Restrictions. *J. Artif. Intell. Res.*, 72: 533–612.

Lundberg, S. M.; and Lee, S. 2017. A Unified Approach to Interpreting Model Predictions. In *Proc. NeurIPS*, 4765–4774.

Marques-Silva, J.; and Ignatiev, A. 2022. Delivering Trustworthy AI through Formal XAI. In *Proc. AAAI*, 12342–12350.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proc. KDD*, 1135–1144.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2018. Anchors: High-Precision Model-Agnostic Explanations. In *Proc. AAAI*, 1527–1535.

Rintanen, J. 2017. Schematic Invariants by Reduction to Ground Invariants. In *Proc. AAAI*, 3644–3650.

Shen, W.; Trevizan, F. W.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proc. ICAPS*, 574–584.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of explanations as model reconciliation. *Artif. Intell.*, 301: 103558.

Steinmetz, M.; Fiser, D.; Eniser, H. F.; Ferber, P.; Gros, T. P.; Heim, P.; Höller, D.; Schuler, X.; Wüstholz, V.; Christakis, M.; and Hoffmann, J. 2022. Debugging a Policy: Automatic Action-Policy Testing in AI Planning. In *Proc. ICAPS*, 353–361.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artif. Intell.*, 168(1-2): 38–69.

Toyer, S.; Thiébaux, S.; Trevizan, F. W.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *J. Artif. Intell. Res.*, 68: 1–68.

Toyer, S.; Trevizan, F. W.; Thiébaux, S.; and Xie, L. 2018. Action Schema Networks: Generalised Policies With Deep Learning. In *Proc. AAAI*, 6294–6301.

Vasileiou, S. L.; Yeoh, W.; Son, T. C.; Kumar, A.; Cashmore, M.; and Magazzeni, D. 2022. A Logic-Based Explanation Generation Framework for Classical and Hybrid Planning Problems. *J. Artif. Intell. Res.*, 73: 1473–1534.

Vinzent, M.; Steinmetz, M.; and Hoffmann, J. 2022. Neural Network Action Policy Verification via Predicate Abstraction. In *Proc. ICAPS*, 371–379.

Younes, H.; and Littman, M. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, CMU.

Yu, J.; Ignatiev, A.; Stuckey, P. J.; Narodytska, N.; and Marques-Silva, J. 2023. Eliminating the Impossible, Whatever Remains Must be True. In *Proc. AAAI*.

Zhang, Z.; and Geißer, F. 2022. Extending Graph Neural Networks for Generalized Stochastic Planning. In *Proc. ICAPS-21 Planning and Reinforcement Learning Workshop (PRL)*.

A Proof of Theorem 1

Before proving Theorem 1, we need one additional definition. Given a partial state s and an action a such that for all $x \in X_s \cap X_{\text{eff}(a)}.\ s[x] = \text{eff}(a)[x]$, we define the *regression* of s through a as the smallest partial state in which applying action a leads to a state satisfying s:

$$\text{reg}_a(s) = (s \ominus \text{eff}(a)) \oplus \text{pre}(a)$$

Lemma 1. *If* $\mathbf{z}$ *explains* $a_1, \ldots, a_n$*, then for all* $i \in \{0, \ldots, n-1\}$*,* $\mathbf{z}_{i+1} := prg_{a_i} \circ \cdots \circ prg_{a_1}(\mathbf{z})$ *explains* a_{i+1}.

We shall prove that for all $i \in \{0, \ldots, n-1\}$, for all state s_{i+1} that completes $\mathbf{z}_{i+1}$, there exists s_i such that i) a_i is applicable in s_i, ii) $s_i \xrightarrow{a_i} s_{i+1}$ is a transition of the planning domain, and iii) s_i completes $\mathbf{z}_i$. By recursion, we end up with a state s_1 that completes $\mathbf{z}_1$ such that $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_i} s_{i+1}$. Since s_1 completes $\mathbf{z}_1 = \mathbf{z}$, we know that $\pi^n(s_1) = a_1, \ldots, a_n$. Therefore, $\pi(s_{i+1}) = a_{i+1}$. As this is true for all s_{i+1}, $\mathbf{z}_{i+1}$ explains a_{i+1}.

The proof will be done by induction, i.e., we assume that it is true for i. (Base case for $i = 0$ is trivially true as $\mathbf{z}_1 = \mathbf{z}$.)

Given our state s_{i+1}, we choose s_i as one of the states that complete $\mathbf{z}_i \oplus \text{reg}_{a_i}(s_{i+1})$, That is

$$s_i \ \supseteq \ \mathbf{z}_i \oplus ((s_{i+1} \ominus \text{eff}(a_i)) \oplus \text{pre}(a_i)) \,. \qquad (2)$$

We prove that s_i satisfies the three points above.
i) Eq. 2 clearly enforces the precondition $\text{pre}(a_i)$, so a_i is indeed applicable in s_i.
ii) We note

$$\text{prg}_{a_i}(s_i) \supseteq (s_{i+1} \ominus \text{eff}(a_i)) \oplus \text{pre}(a_i) \oplus \text{eff}(a_i).$$

which can be simplified by

$$\text{prg}_{a_i}(s_i) \supseteq s_{i+1} \oplus \text{pre}(a_i) \oplus \text{eff}(a_i). \qquad (3)$$

We also note that all variables $x \in X$ are specified in $\text{prg}_{a_i}(s_i)$, regardless of the choice of s_i. Consider any variable x; does $\text{prg}_{a_i}(s_i)[x] = s_{i+1}[x]$ hold?

- If $x \in X_{\text{eff}(a_i)}$, then $\text{prg}_{a_i}(s_i)[x] = \text{eff}_{a_i}(x)$ and, since $\mathbf{z}_{i+1} = \text{prg}_{a_i}(\mathbf{z}_i)$, $s_{i+1}[x] = \mathbf{z}_{i+1}[x] = \text{eff}_{a_i}(x)$.
- If $x \in X_{\text{pre}(a_i)} \backslash X_{\text{eff}(a_i)}$, then $\text{prg}_{a_i}(s_i)[x] = s_i(x) = \text{pre}(a_i)[x]$ since a_i does not modify x and a_i is applicable in s_i.
  Furthermore, we note that s_{i+1} completes $\text{prg}_{a_1}(\mathbf{z}_i)$ and that $\mathbf{z}_i[x] = \text{pre}(a_i)[x]$ since (by induction) $\mathbf{z}_i$ explains a_i; therefore $s_{i+1}[x] = \text{pre}(a_i)[x] = \text{prg}_{a_i}(s_i)[x]$.
- If $x \in X \backslash X_{\text{pre}(a_i)} \backslash X_{\text{eff}(a_i)}$, then $\text{prg}_{a_i}(s_i)[x] = s_i[x] = s_{i+1}[x]$.

So $\text{prg}_{a_1}(s_i) = s_{i+1}$.
iii) Does s_i complete $\mathbf{z}_i$? Since, by construction, s_i completes $\mathbf{z}_i \oplus ((s_{i+1} \ominus \text{eff}(a_i)) \oplus \text{pre}(a_i))$, the question is whether some assignment in the right operand of $\mathbf{z}_i \oplus$ contradicts an assignment in s_i. We know that $\mathbf{z}_i$ explains a_i, so pre_{a_i} will not contradict $\mathbf{z}_i$. State s_{i+1} completes $\mathbf{z}_{i+1}$ which differs with $\mathbf{z}_i$ only on $\text{eff}(a_i)$. However, the expression above removes $\text{eff}(a_i)$ from s_{i+1}, so that the right operand does not map any variable to a different value than $\mathbf{z}_i$.

Lemma 2. *Let* $\mathbf{z}$ *be a partial state, and for all* $i \in \{0, \ldots, n-1\}$*,* $\mathbf{z}_{i+1} := prg_{a_i} \circ \cdots \circ prg_{a_1}(\mathbf{z})$*. If for all i,* $\mathbf{z}_i$ *explains* a_i*, then* $\mathbf{z}$ *explains* $a_1, \ldots, a_n$.

Assume that $\mathbf{z}_i$ explains a_i for all i. Let s be a state completing $\mathbf{z}$ and let $s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n$ be the trajectory obtained by applying the sequence of actions $a_1, \ldots, a_n$ from $s_1 = s$. We shall prove $\pi(s_i) = a_i$ for all i (which proves that π recommends $a_1, \ldots, a_n$); this is proven by showing that s_i completes $\mathbf{z}_i$.

The state s_{i+1} is defined as $\text{prg}_{a_i} \circ \cdots \circ \text{prg}_{a_1}(s)$. Do we have $\forall x \in X_{\mathbf{z}_{i+1}}.\ s_{i+1}[x] = \mathbf{z}_{i+1}[x]$? Let j be the largest index in $\{1, \ldots, i\}$ such that $x \in X_{\text{eff}(a_j)}$. If j does not exist, then $s_{i+1}[x] = s[x] = \mathbf{z}[x] = \mathbf{z}_{i+1}[x]$. Otherwise, $s_{i+1}[x] = \text{eff}(a_j)[x] = \mathbf{z}_{i+1}[x]$. Either way, the variables of s_{i+1} map to the same value as those of $\mathbf{z}_{i+1}$. Therefore, $\mathbf{z}_{i+1}$ explains a_{i+1} in s_{i+1}.

Theorem 1 is a consequence of Lemmas 1 and Lemma 2.