Explainable Representation of Finite-Memory Policies for POMDPs using Decision Trees

Muqsit Azeem
Technical University of Munich
Munich, Germany
muqsit.azeem@tum.de

Debraj Chakraborty
Nanyang Technological University
Singapore
debraj.chakraborty@ntu.edu.sg

Sudeep Kanav
Masaryk University
Brno, Czech Republic
kanav@fi.muni.cz

Jan Kretinsky
Masaryk University
Brno, Czech Republic
jan.kretinsky@fi.muni.cz

ABSTRACT

Partially Observable Markov Decision Processes (POMDPs) are a fundamental framework for decision-making under uncertainty but often require infinite memory, making implementation infeasible and many problems undecidable. While finite-memory policies provide a practical alternative, they remain complex and challenging to interpret.

To address this, we propose a novel *representation* of finite-memory policies that is both (i) interpretable and (ii) smaller, enhancing explainability without sacrificing optimality. To that end, we combine Mealy machines and decision trees (DTs); the latter describing simple, stationary parts of the policies and the former describing how to switch among them.

We design a translation for finite-state-controller (FSC) policies from standard literature into our new representation, enhancing explainability and compactness while preserving optimality. Notably, our method seamlessly generalizes to other variants of finite-memory policies. Additionally, we identify unique properties of "attractor-based" policies, enabling the construction of even smaller, simpler representations. Finally, through multiple case studies, we illustrate the improved explainability and practicality of our approach.

KEYWORDS

Explainability, POMDP, Decision Trees, Finite-State Controllers

1 INTRODUCTION

POMDP and FSC. Partially observable Markov decision processes (POMDPs) provide a foundational framework for decision-making under uncertainty, where the agent lacks complete information about the current state. They have applications ranging from autonomous robotics and intelligent assistants [24] to healthcare systems [27]. Unfortunately, decision-making in POMDPs is inherently complex. In particular, infinite memory may be required for optimality for many objectives. Not only are such policies impractical for real-world implementation, but this complexity also makes finding the optimal policies particularly challenging and, in most settings, undecidable [37]. Consequently, finite-memory policies are often considered, e.g. [2, 3]. Besides rendering various problems decidable (or at least allowing for efficient heuristics), they are clearly more amenable to implementation than general, infinite-memory policies. Finite-memory policies are formalized as *finite-state controllers (FSCs)*, essentially Mealy machines (finite-state automata with both input and output) where each state corresponds to a stationary (a.k.a. memoryless) policy and the automaton transitions describe how to switch among them. However, while FSCs make POMDP policies implementable, certain issues arise concerning their explainability, which we discuss below and which motivate the need for more explainable representations of FSCs.

Explainability. In many real-world domains—from autonomous driving to healthcare decision support—opaque or complex policies pose critical risks. Black-box policies can be hard to validate or trust, especially under uncertainty. Hence, explainable controllers are not merely preferable but also essential from the legal perspective [1]. The automata representation is often regarded as interpretable and even explainable [10] due to its graphical structure and typically low number of states. However, this issue is more complex as the automaton drawing does not depict the whole policy, notably: (i) Each state corresponds to a stationary policy, which is typically a *table* with an action to be played for each observation; due to its often enormous size it is hard to explain and thus does not expose the goal of that particular policy. (ii) Each transition between states is typically labeled by many observations (often inconceivably many due to the combinatorial explosion), effectively hiding the reasons to switch to another policy. While both issues prevent explainability, the latter one is more severe due to two reasons. First, the size of the representation of transitions is even greater than that of the policies. Second, while the former issue of representing a stationary policy has been tackled in literature, e.g. [7], the latter issue of explainable transitions not so. To summarize, the issues (i) and (ii) often obscure the decision logic: large action tables obscure the intent of each policy, and opaque transition labels conceal why the controller switches states. This lack of clarity undermines trust, especially when certification or debugging is required.

We address this lack of transparency systematically in several steps, exploiting the factored structure of states and observations.
• *First*, we translate the tabular representations of the stationary policy in each FSC state into a decision tree (DT). While this has been done neither for POMDP, nor for randomizing policies, the generalizations are straightforward and thus a mere application of methods from the literature.

• *Second*, while history-dependent policies have never been represented by DTs, we propose to do so by combining the FSC structure and the transition labels (being sets of observations) translated into DTs. While an alternative approach might be to encode the labels as Boolean formulae (over the observations or their variables), DT have the advantage of their hierarchical structure. Indeed, while a formula (or its syntax tree) does not reflect the importance of each part, DT features its most important decisions close to the root with the less important ones near the leaves. This allows for cheap and transparent pruning (ignoring the less important layers of the tree), offering a trade-off between precision and explainability. In particular, in order to understand when stationary policies switch, one can easily understand the frequent simpler cases first and subsequently contemplate also the further corner-cases.

To summarize the two points, this translation from FSC to "DT-FSC" replaces opaque action and transition tables with interpretable decision trees that expose how observations guide both action selection and memory updates. While making the representation more concise (and interpretable due to the factored structure), we retain the exact functional equivalence to the original policy. On top of that, searching the tables is replaced by evaluating the decision trees, which are isomorphic to if-then-else code, and can thus be executed orders of magnitude *faster* [33].

• *Third*, we profit from the new DT-FSC representation, in order to provide an *explanation* of the policy. On the one hand, prior work on explainable policies via DT [6, 7] adopts the widely used view that *the size of the DT* is a practical proxy for explainability: smaller trees are easier to inspect, understand, and trust. This perspective is further supported by recent studies in human-robot interaction and decision-making tasks, e.g. [45]. Consequently, we adopt this proxy of explainability for the individual trees. On the other hand, we have to critically assess this perspective when it comes to history-dependent policies since the sequential, dynamic changes in behaviour pose an inherent additional challenge for explainability. We scrutinize the resulting structure in the light of several case studies. To that end, we cast away quantitative proxies of explainability and observe how our result stands the ultimate (however subjective) test of reaching the "aha, now I see"-moment. We have observed the following: (i) In contrast to the large FSC, from the DT-FSC representation one can trivially read off several English sentences describing completely the rules of the policy. (ii) This in turn does *not* itself constitute an intuitive explanation of the policy behaviour to be understood by a human, e.g. a colleague, mostly due to the sequential composition of varying behaviours. (iii) An intuitive explanation can be derived in each case, using manual effort. For instance, the rules "in the initial configurations we move to the south" and"the direction at some point changes to the east, then to the north, then to the west" must be manually turned into "we are going in circles". (iv) Most importantly, while the DT-FSC representation allowed for the manually gained insight, the original tabular FSC with its sheer size and non-factored, structureless information did not. We conclude that our new DT-FSC can serve as the first technical step towards generating actual explanations of history-dependent policies.

Our technical contribution can be summarized as follows: In particular, we propose a new data structure to represent FSCs (history-dependent policies), merging the advantages of the Mealy machines and the interpretable *decision trees* (DTs). The latter is used not only for representing the stationary policies similarly to literature, e.g. [7], but importantly also for the automata transitions. This reduces the redundancy by grouping related observations, which together with the DT structure yields a panoramic view of decision paths of conceivable size, and highlights critical variables and decision points. Besides, we identify specific properties of "attractor-based" policies [30], which allow us to construct yet simpler and smaller representations with more compact and fewer transitions. Notably, *our transformation procedure alters only the representation obtained from existing tools, preserving both the policies and their optimality.* Further, as it is *modularly* defined on parts of the original policy, it captures the parts separately, simplifying the explanation compared to a larger monolith. As a result, lowering the barrier for their human inspection, validation, and debugging moves us closer to making formally verified decision-making systems both transparent and usable in practice.

Related Work. **POMDPs** can be solved using various exact [25, 44] or simulation-based [43] techniques. Formal verification tools like Storm [29] use *belief*-based analysis [8] to find an optimal policy. For qualitative objectives such as almost-sure reachability, SAT/SMT-based methods (both one-shot [20] and iterative approach [30]) are shown to be effective. We build on these approaches by transforming their FSC outputs into more interpretable forms. Our technique can take specific advantage of this iterative SMT-based approach.

FSCs representing POMDP policies have been synthesized by inductive synthesis approaches of Paynt [4], learnt by using L^* [10], extracted from recurrent policy networks [18, 34] or computed using a combination of belief-based and inductive approaches together [2]. Our technique takes these types of controllers as input and generates their more explainable representation. Another relevant model is the *plan graph* [31]: a Moore machine FSC where the nodes are labeled by actions and edges are labeled by observations. In our Mealy-machine FSC, the observation-action mapping is explicit on each edge, making it easier to read and audit "if observe o, then take a and go to the next stationary policy" without mentally combining node and edge labels. Further, Mealy controllers are generally smaller, e.g., a policy can be a one-node Mealy controller but generally needs $|A|$ nodes as a policy graph.

Explainability in Decision Making. Explainability in sequential decision making has been explored through various lenses in the AI community. Prior works include policy summarization [28, 32], counterfactual reasoning [38], and causal models [42], which seek to make policy decisions understandable through simplification or causal explanation. More generally, explainability is often defined as the degree to which a human can understand the cause of a decision [40], or trace the steps taken by an agent [26] in a black-box setting. In our work, we (i) adopt the framework suggesting small decision trees as means to explain a controller at one glance (rather than a single decision) [5–7, 13, 16, 33], and (ii) sanity-check its compatibility with explanations being a few English sentences triggering the aha moment. (For the sake of readability, we do not drag the reader into the details of cognitive complexity [17] to define the latter more formally.)

Decision Trees (DTs) provide a natural mechanism for achieving explainability in this setting. Their hierarchical, rule-based structure aligns closely with human reasoning and supports transparent analysis of decision logic [6, 7]. DTs have been widely used in data mining [39], supervised learning [15], and controller synthesis, including the representation of controllers in different systems [12, 14, 41]. The dtcontrol tool [6] has demonstrated that stationary policies can be compactly and effectively represented using DTs. Building on this, our work proposes the use of DTs beyond stationary policies to finite-memory controllers through a modular transformation that embeds interpretability directly into the policy representation. By representing both action selection and memory transitions as decision trees, our approach produces a globally explainable controller without compromising correctness or optimality.

Technically similar structures, such as decision diagrams (BDDs), have also been used to represent transitions and output in symbolic automata [23]. However, despite being more compact in some settings, they have a few disadvantages in this context. In particular, their interpretability is hindered by non-Boolean variables being converted to bitvectors using bit-blasting; they lack variability of ordering in different branches, which is important for controllers; they handle don't care inputs in less efficient ways than DT. In contrast, more interpretable DT are easier for humans to follow, and are compatible with tools like dtControl [7], which support expert-in-the-loop pruning and analysis. Due to these and further reasons they have not been used as much as DT [6], although the recent work [19] diminishes the gap.

2 PRELIMINARIES

A *probability distribution* on a countable set S is a function $d : S \to [0, 1]$ such that $\sum_{s \in S} d(S) = 1$. *Support* of a probability distribution d, is the set $\text{supp}(d) = \{s \in S \mid d(s) > 0\}$. We denote the set of all probability distributions on the set S as $\text{Dist}(S)$.

Definition 1 (POMDP)**.** A *partially observable MDP (POMDP)* is a tuple $\mathcal{P} = (S, A, P, R, s_{\text{init}}, Z, O)$ where S is a countable set of states, A is a finite set of actions, $P : S \times A \rightharpoonup \text{Dist}(S)$ is a partial transition function, $R : S \times A \to \mathbb{R}$ is a reward function, and $s_{\text{init}} \in S$ is the initial state, Z is a finite set of observations, and $O : S \to Z$ is an observation function[1] that maps each state to an observation.

We assume that the observation space is factored, i.e. $Z = \prod_{i=1}^{k} Z_i$ where each component Z_i represents a distinct feature of the observation, meaning any observation $z \in Z$ can be represented as a tuple of features $(z_1, z_2, \ldots, z_k)$, where each $z_i \in Z_i$. Note that, this assumption is quite natural. In many real-world scenarios, observations come from a set of independent components, each reflecting a specific aspect of the state. For example, in robotics, the observation can include the robot's position, battery level, and different sensor readings.

We define the set of *enabled actions* in $s \in S$ by $A(s) = \{a \in A \mid P(s, a) \text{ is defined}\}$. For a POMDP $\mathcal{P}$, a *path* is a sequence $s_0 a_0 s_1 a_1 s_2 \ldots$ such that for all i, $a_i \in A(s_i)$ and $P(s_i, a_i)(s_{i+1}) > 0$.

[1]Deterministic observation functions are sufficient as the probabilistic aspect of the observation function can be encoded in the transition function [21]. Thus our definition with a deterministic observation function does not lose generalization, but rather, simplifies the notation and also make it more explainable.

We extend the notion of observations to the paths in a natural way: for a path $\rho = s_0 a_0 s_1 a_1 \cdots$, $O(\rho)$ is the sequence $O(s_0) a_0 O(s_1) a_1 \cdots$. A *history* is a finite sequence of observations and actions $h = o_0 a_0 \ldots a_{i-1} o_i$ such that there exists a finite path $\rho = s_0 a_0 s_1 \ldots s_i$ with $o_j = O(s_j)$ for $0 \le j \le i$. We denote the set of all histories of $\mathcal{P}$ by $\text{Hist}_{\mathcal{P}}$.

Definition 2 (Policy for POMDPs)**.** A *policy* for a POMDP $\mathcal{P}$ is a function $\pi : \text{Hist}_{\mathcal{P}} \to \text{Dist}(A)$ mapping histories to actions.

A policy π is *deterministic* if $|\text{supp}(\pi(h))| = 1$ for all paths $h \in \text{Hist}_{\mathcal{P}}$. Otherwise, it is *randomized*. A policy π is called *stationary* or *memoryless* if it depends only on the last observation.

Policies resolve nondeterminism by converting a POMDP to a fully observable Markov chain. This Markov chain induces a unique probability measure $\mathbb{P}_{\pi, s_0}$ over paths starting in initial state s_0 [9].

Policy Synthesis. The *policy synthesis* problem consists of finding a policy that satisfies a certain objective φ. In this paper, we consider infinite-horizon reachability or expected total reward objectives. Given a set of target states $T \subseteq S$, we denote $\mathbb{P}_{\pi, s_0}(\Diamond T)$ to be the probability of reaching a state in T following the policy π. In the case of *almost-sure* reachability objective, a policy π is *winning* from s_0 if $\mathbb{P}_{\pi, s_0}(\Diamond T) = 1$. In this case, we denote $\mathbb{E}_{\pi, s_0}(\text{reward}(\Diamond T))$ to be the expected total reward accumulated till we reach the states in T. Alternatively, we can look at *quantitative* objectives: given an objective φ and a threshold $\lambda \in (0, 1)$, a policy is winning if $\mathbb{P}_{\pi, s_0}(\Diamond T) \ge \lambda$.

Finite-State Controller (FSC). We provide an abstract definition of FSC and then give instantiations used in literature.

Definition 3 (FSC)**.** A finite state controller (FSC) is a tuple $\mathcal{F} = (N, n_{\text{init}}, \delta, \gamma)$ where N is a finite set of nodes, $n_{\text{init}} \in N$ is the initial node, $\delta \colon N \times \mathit{Obs}\mathit{Dom}_{\delta} \to N$ is the transition function, $\gamma \colon N \times \mathit{Obs}\mathit{Dom}_{\gamma} \to A$ is the action mapping, where $\mathit{Obs}\mathit{Dom}_{\delta}$ and $\mathit{Obs}\mathit{Dom}_{\gamma}$ are observation domains over Z.

Typically, in an FSC, $\mathit{Obs}\mathit{Dom}_{\gamma} = Z$ and $\mathit{Obs}\mathit{Dom}_{\delta} = Z$. Different works in the literature adopt slightly different notions of these observation domains. For instance, some definitions let transitions depend only on the current observation [10], i.e., $\mathit{Obs}\mathit{Dom}_{\delta} = Z$, while others [2, 30] use both the current and the next (posterior) observation, i.e., $\mathit{Obs}\mathit{Dom}_{\delta} = Z \times Z$. In this case, in state s with observation z, an agent following an FSC $\mathcal{F}$ executes the action $a = \gamma(n, z)$ and then based on the current and the next observation z and $z' = O(s')$, the FSC evolves to node $n' = \delta(n, z, z')$.

Remark 1. Our formulation abstracts over these choices by introducing the generic domains $\mathit{Obs}\mathit{Dom}_{\gamma}$ and $\mathit{Obs}\mathit{Dom}_{\delta}$. This unifies the variants used in the literature [2, 10, 30], and our framework can handle controllers defined under any such convention.

Definition 4 (Decision Tree (DT))**.** Given an input space $\mathbb{X}$ and an output space $\mathbb{A}$, a *decision tree* $T : \mathbb{X} \to \mathbb{A}$ is a binary tree where each inner node v is labeled with a *predicate* $P_v : \mathbb{X} \to \{\texttt{true}, \texttt{false}\}$. Each edge corresponds to an outcome of the predicate: solid for `true`, dotted for `false`. Each leaf ℓ is labeled with an output $a_\ell \in \mathbb{A}$. The output $T(x)$ for an input $x \in \mathbb{X}$ is determined by evaluating predicates from the root to a leaf, and returning the output label a_ℓ of that leaf.

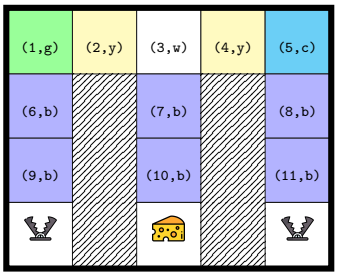


Figure 1: Maze with state-observation labels; colors indicate observations: green (g), yellow (y), white (w), cyan (c), and blue (b). Shaded cells are inaccessible.

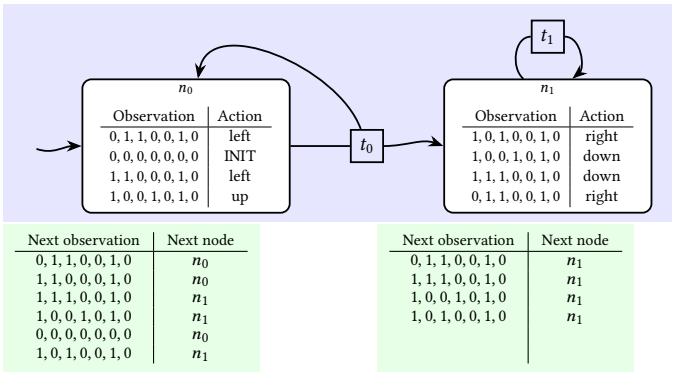


Figure 2: An FSC represented with explicit tables. Each observation is encoded as a vector over the seven observale variables. The transition tables t_0 and t_1 are below nodes n_0 and n_1, respectively. In this example, only the posterior observations are relevant to make decisions, so only the posterior observations (z') are shown in the tables.

Decision Trees for Policy Representation. DT learning algorithms can exploit the structure of factored observations to represent policies in a compact and explainable way. A stationary policy can be exactly represented by a DT [6], where the input is the state set S and the output labels correspond to actions A suggested by the policy. Unlike conventional DTs in machine learning, which aim to generalize and may tolerate classification errors, DTs in this context must represent the policy exactly to ensure correctness. The following sections extend this idea to non-stationary policies.

3 AN ILLUSTRATIVE EXAMPLE EXPLAINING POLICY REPRESENTATION

We illustrate how a non-stationary POMDP policy can be represented as an FSC. As a running example, we use the cheese-maze model [36] as detailed in Example 1.

Example 1 (Maze)**.** Consider the maze in Fig. 1, modeled as a POMDP. A mouse is initially placed randomly in the grid in a state with observation blue. Its objective is to reach the target (cheese) while avoiding the traps. Each cell represents a distinct state, and the mouse can move in one of the four possible cardinal directions at each step unless blocked by a wall. However, the mouse's observations are limited to local wall configurations, making it unable to distinguish between cells with identical surroundings (indicated by the same color). Consequently, there are only seven distinct observations, with the target and traps considered as separate ones.

Finite State Controller Representation. In Example 1, the observation space is structured with the following boolean variables: `CanGoDown`, `CanGoLeft`, `CanGoRight`, `CanGoUp`, which indicate if the mouse can move in each direction. Additionally, there are three more observable variables: `bad` to denote whether the mouse is in a trap, `clk` to denote if the mouse has entered the maze and `goal`, which indicates whether the mouse is at the target. Fig. 2 shows an FSC representation of a winning policy. In the tables, each observation is represented as a vector over these seven variables.

For each node of the FSC, there are two associated tables: (i) an *action table* that specifies which action to take for each observation, and (ii) a *transition table* that determines the next node based on the current and next observation. At each step, the mouse observes its environment and selects an action according to the action table of the current node. After executing the action and receiving a new observation, the mouse may transition to a new controller node (e.g., from n_0 to n_1) based on the transition table. Each node thus defines a stationary policy that the mouse follows while its controller remains in that node. *This mechanism allows the FSC to encode finite memory: the current node implicitly tracks part of the observation history.*

Decision-Tree-Based Explainability. Instead of using large tables to specify the stationary policy and transitions at each node of FSC, we translate both into decision trees, resulting in a DT-FSC (see Fig. 3). The DT-FSC in Fig. 3 has two nodes, each with one DT for action selection and one for transition between the FSC nodes. While the FSC nodes still capture history, the DTs exploit the structure in observations to make the representation more compact and interpretable. For instance, the predicate `CanGoUp` is absent from all DTs, showing it plays no role in decision making. Thus, complementing FSCs with DTs provides a clearer and more structured explanation than policy tables.

In this example, the mouse starts with initial policy in node n_0. If it is not yet placed in the grid, it first plays `INIT`, which randomly places it in a state with observation blue. From there, it moves `up` on blue and `left` on yellow or cyan. It remains in node n_0, effectively waiting until the next (posterior) observation becomes green or white—at which point it switches to the policy in node n_1. The policy in n_1 directs the mouse to go `right` when it sees green or yellow, continuing until it reaches white. On white or blue, it then suggests moving `down` to eventually reach the cell containing the cheese (see Fig. 6 in Section A.1 for a pictorial representation). Note that conflicting behaviors for observations like blue and yellow are resolved through memory: the FSC node captures this context.

For smaller grid-based examples, such as this one, a pictorial policy representation may be easier to understand. However, the DTs provides a general, domain-independent representation. Moreover, while the grid policy size grows linearly with the problem, the DT-based representation may remain compact if the decision structure stays simple.

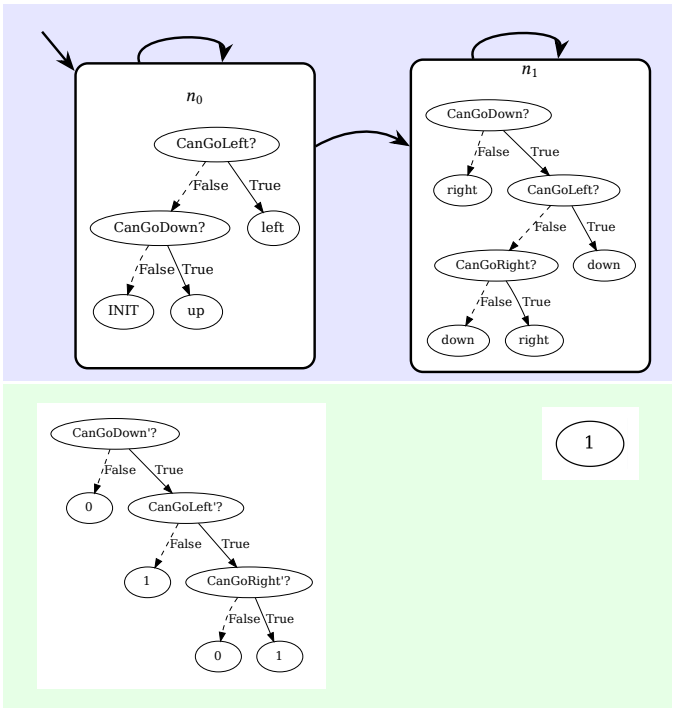


Figure 3: Explaining FSCs using DTs. For an observation variable z, z' would denote the next observation, *i.e.*, the observation the mouse sees after taking the action. A DT in a node describes a positional policy, a DT below a node describes the next node for the next observation (curved arrows in the FSC indicate possible transitions). Here, the solid edges in the DT are used to denote `true`, dotted to denote `false`.

4 EXPLAINABLE REPRESENTATION OF FSCS VIA AMALGAMATION WITH DTS

In this section, we formally describe how explainability of FSCs can be enhanced using DTs by introducing a specific data structure we call *DT-FSC*. We then discuss its general applicability and further describe a particular scenario where FSCs (and their corresponding DT-FSCs) can be built using an iterative approach, which simplifies the explanation process. For this scenario, we also introduce a specialized data structure that offers the same level of explainability in a more compact form.

Definition 5 (DT-FSC)**.** Given an FSC $\mathcal{F} = (N, n_{\text{init}}, \delta, \gamma)$, we define its DT-FSC representation as follows:

$$\mathcal{F}_{\text{DT}} = (N, n_{\text{init}}, \{\text{DT}_{\delta,n}\}_{n\in N}, \{\text{DT}_{\gamma,n}\}_{n\in N}),$$

where, for each $n \in N$, $\text{DT}_{\delta,n}$ is the decision tree representation of δ_n, and $\text{DT}_{\gamma,n}$ is the decision tree representation of γ_n.

We define the semantics of a DT-FSC via a single-step execution, analogous to the single-step execution of an FSC, where δ and γ are replaced by $\text{DT}_{\delta,n}$ and $\text{DT}_{\gamma,n}$, respectively.Fig. 3 shows the DT-FSC representation of the policy for the FSC in Fig. 2. The trees inside the nodes describe the stationary policies. The trees below the node describe the transition between the nodes. Although the transition function formally depends on both current and next observations, the learned decision trees simplify this by ignoring irrelevant inputs. For example, the transition DT in Fig. 3 relies only on the posterior observation, even though the original transition function considers both.

Algorithm 1: Translation from FSC to DT-FSC

Input: FSC $\mathcal{F} = (N, n_{\text{init}}, \delta, \gamma)$
Output: DT-FSC $\mathcal{F}_{\text{DT}} = (N, n_{\text{init}}, \{\text{DT}_{\delta,n}\}_{n\in N}, \{\text{DT}_{\gamma,n}\}_{n\in N})$

1 **for** $n \in N$ **do**
  // Phase 1: Dataset construction
2 Construct dataset $\text{D}_{\gamma,n}$ from γ;
3 Construct dataset $\text{D}_{\delta,n}$ from δ;
  // Phase 2: Decision tree learning
4 $\text{DT}_{\gamma,n} \leftarrow$ `LearnDT`$(\text{D}_{\gamma,n})$;
5 $\text{DT}_{\delta,n} \leftarrow$ `LearnDT`$(\text{D}_{\delta,n})$;
6 **return** $\mathcal{F}_{\text{DT}} = (N, n_{\text{init}}, \{\text{DT}_{\delta,n}\}_{n\in N}, \{\text{DT}_{\gamma,n}\}_{n\in N})$;
7 ——— Helper: `LearnDT` ———
8 **Function** `LearnDT` (D):
9 **if** *all* $(x, y) \in$ D *have the same label* y **then**
10 **return** `Leaf`(y);
11 **else**
12 $p \leftarrow$ `FindBestPredicate`(D);
13 $\text{D}_{left} \leftarrow \{(x, y) \in \text{D} \mid p(x)\}$;
14 $\text{D}_{right} \leftarrow \{(x, y) \in \text{D} \mid \neg p(x)\}$;
15 **return** `Node`$(p,$ `LearnDT`$(\text{D}_{left}),$ `LearnDT`$(\text{D}_{right}))$;

4.1 From FSC to DT-FSC: Translation Algorithm

Algorithm 1 outlines the translation of an FSC into a DT-FSC in two phases: (1) dataset construction and (2) decision tree learning.

(1) Dataset Construction. For each node $n \in N$ in the FSC $\mathcal{F} = (N, n_{\text{init}}, \delta, \gamma)$, we extract two functions: i) $\gamma_n \colon \mathit{ObsDom}_\gamma \to A$ where $\gamma_n(o_\gamma) = \gamma(n, o_\gamma)$, and ii) $\delta_n \colon \mathit{ObsDom}_\delta \to A$ where $\delta_n(o_\delta) = \delta(n, o_\delta)$. A function $f \colon X \to Y$ with a finite domain X can be treated as a finite set $\text{D} \subseteq X \times Y$, a dataset of feature-label pairs. We obtain $\text{D}_{\gamma,n}$ from γ_n and $\text{D}_{\delta,n}$ from δ_n.

(2) Decision Tree Learning. For $n \in N$, we learn the decision trees $\text{DT}_{\gamma,n}$ and $\text{DT}_{\delta,n}$ representing the mapping γ_n and δ_n exactly. A DT learning algorithm recursively splits the dataset to minimize the entropy of the resulting subsets. The entropy of a dataset D over labels Y is defined as $\text{entr}(\text{D}) = -\sum_{y\in Y} p_y \log(p_y)$ where p_y is the fraction of examples labeled y, i.e. $p_y = \frac{|(x,y)\in\text{D}|}{|\text{D}|}$. The best split is chosen via `FindBestPredicate` (line 12 of `LearnDT` helper function) as the one that minimizes the total entropy after the split: $\text{entr}(\text{D}_{left}) + \text{entr}(\text{D}_{right})$. Since our goal is exact representation, we recurse until all examples in a subset share the same label. The resulting DT-FSC $\mathcal{F}_{\text{DT}}$ preserves the original FSC behavior exactly while offering a more interpretable representation.

4.1.1 General Applicability of our DT-FSCs. While optimal policies for POMDPs may require infinite memory in general [37], in practice, "good enough" finite-memory policies are considered to balance expressiveness with tractability [2, 10]. For instance,

[2] uses an anytime inductive synthesis approach that iteratively searches for optimal FSCs of increasing sizes. [10] uses Storm to find a policy and then learns an FSC from it. Our DT-FSC translation framework is broadly applicable to such settings.

Storm uses partial exploration [11] and produces a randomized policy for the unexplored regions, referred to as *cutoffs*. Even in such cases, we can construct a DT-FSC that explains both the stationary policy and the transition function for each node. Since the number of cutoffs is finite, the policy suggests only finitely many randomized actions—allowing us to build DTs with a finite number of leaf labels.

Remark 2. *The performance of our DT-FSC policy remains the same as the one of the input policy. In particular, the action and transition functions are total, and the DTs are trained to overfit the data, guaranteeing functional equivalence with the input FSC, as done in [6].*

4.2 skip-FSC Optimization for Attractor-based Policies (for Almost-sure Reachability)

For almost-sure reachability objectives, we construct a policy to reach the target by iteratively computing the attractor[2] of the winning region (target set), following the approach of [30]. For such policies, we provide an even more compact representation. Note that, here we have $Obs\mathrm{Dom}_\gamma = Z$ and $Obs\mathrm{Dom}_\delta = Z \times Z$. Starting from target states, in each iteration, an observation-based stationary policy is obtained that either reaches the target or reaches an observation from which we have already found a winning policy in a previous iteration. In the second case, the policy would switch to the previously calculated winning policy. This iterative procedure also helps us construct the FSC representing a winning policy. At iteration i, we add a new node n_i to the FSC. The policy σ_i found in the iteration would describe the action mapping, *i.e.*, $\gamma(n_i, z) = \sigma_i(z)$. If the policy requires switching to a previously computed policy, this is captured by the transition function δ. This can be conceptualized as follows:

Case 1: Based on the current observation z, the policy can suggest an action a and does not switch to any other policy. In that case, in the constructed FSC, $\gamma(n_i, z) = a$ and the node of the FSC would not change.

Case 2: Upon encountering an observation z', the policy can switch to a previously computed policy σ_j (without taking any action). In that case, in the constructed FSC, for all $z \in Z$, we would have $\delta(n_i, z, z') = n_j$, where n_j is the node corresponding to the policy σ_j added in the j^{th} iteration.

Case 3: From observation z, the policy takes an action a and then depending on the subsequent observation z', it can switch to another policy σ_j. In that case, in the constructed FSC, for all $z' \in Z$, we would have $\delta(n_i, z, z') = n_j$, where n_j is the node corresponding to the policy σ_j added in the j^{th} iteration. We also have $\gamma(n_i, z) = a$.

The iterative procedure concludes once a winning policy is identified from the initial observations. The controller created by this approach would still have some nodes non-reachable from the initial node. We remove these irrelevant nodes by graph post-processing.

[2]An attractor is the set of observations from which a policy can ensure reaching the target region, possibly in multiple steps.

Algorithm 2: Semantics of skip-FSC: single-step execution

Input: skip-FSC $\mathcal{F}_{\text{skip}} = (N, n_{\text{init}}, \delta_{\text{skip}}, \gamma_{\text{skip}})$, POMDP $\mathcal{P} = (\mathcal{M}, Z, O)$, current node n and observation z

repeat
  $a \leftarrow \gamma_{\text{skip}}(n, z)$;
  $n \leftarrow \delta_{\text{skip}}(n, z, z)$;
until $a \neq skip$;
Execute a and go to the next state with observation z';
$n \leftarrow \delta_{\text{skip}}(n, z, z')$;

For more details of the algorithm, see [30]. Here we will give a bit more intuition about case 2. Suppose at iteration i, we find a policy that is winning from any state with observation z. Then at any later iteration, $j > i$, upon observing z, the agent can just switch to the previously discovered policy that is winning from z. This gives us the following fact:

Observation 1. *For any winning observation $z \in Z$, there exists an iteration i_z where we find a policy that is winning from any state with observation z. Then in the FSC generated by the iterative approach, for any $j > i_z$, for any $z' \in Z$, $\delta(n_j, z', z) = n_{i_z}$.*

Skip Transitions and skip-FSC. We introduce an alternative representation for the FSCs produced by the iterative approach, referred to as skip-FSCs where we exploit the structural characteristics as described in Observation 1. We define a skip-FSC as follows:

Definition 6 (skip-FSC)**.** A skip-FSC is a tuple $\mathcal{F}_{\text{skip}} = (N, n_{\text{init}}, \delta_{\text{skip}}, \gamma_{\text{skip}})$ where N is a finite set of nodes, $n_{\text{init}} \in N$ is the initial node, $\delta_{\text{skip}} : N \times Z \times Z \to N$ is a special transition function, and $\gamma_{\text{skip}} : N \times Z \to A \cup \{\text{skip}\}$ is a special action labeling.

In state s, an agent receives observation $z = O(s)$. At current node n, if $\gamma_{\text{skip}}(n, z) = \text{skip}$, the agent transitions to the node $n' = \delta_{\text{skip}}(n, z, z)$ and continues this process until reaching a node n'' with $\gamma_{\text{skip}}(n'', z) \neq \text{skip}$. The agent takes the action $a = \gamma_{\text{skip}}(n'', z)$. The state of the MDP is then changed to s' according to the probability distribution. Then, based on the current and the next observation z and $z' = O(s')$, the FSC updates to the new node $\delta_{\text{skip}}(n'', z, z')$. The semantics of this FSC is described in Algorithm 2.

4.2.1 Translation to a skip-FSC. Given an FSC $\mathcal{F} = (N, n_{\text{init}}, \delta, \gamma)$ satisfying Observation 1, we translate to a skip-FSC $\mathcal{F}_{\text{skip}} = (N, n_{\text{init}}, \delta_{\text{skip}}, \gamma_{\text{skip}})$ as described in Algorithm 3. The algorithm iterates over all win-

Algorithm 3: Attractor-based FSC Policy to skip-FSC

Input: Attractor-based FSC $\mathcal{F} = (N, n_{\text{init}}, \delta, \gamma)$
Output: skip-FSC $\mathcal{F}_{\text{skip}} = (N, n_{\text{init}}, \delta_{\text{skip}}, \gamma_{\text{skip}})$

$\delta_{\text{skip}} \leftarrow \delta$; $\gamma_{\text{skip}} \leftarrow \gamma$;
for *winning observation* z' **do**
  for $j > i_{z'}$ *and* $z \in Z$ **do**
    Remove $(n_j, z, z') \mapsto n_{i_{z'}}$ from δ_{skip};
    for $j > k > i_{z'}$ **do**
      $\delta_{\text{skip}}(n_k, z, z') \leftarrow n_{k-1}$;
      $\gamma_{\text{skip}}(n_k, z) \leftarrow \text{skip}$;
return $\mathcal{F}_{\text{skip}} = (N, n_{\text{init}}, \delta_{\text{skip}}, \gamma_{\text{skip}})$

ning observations z', modifying the transitions and action labels to incorporate the skip mechanism. Specifically, for each node n_j with index greater that $i_{z'}$ (defined in Observation 1), the direct switching to $n_{i_{z'}}$ is removed and replaced with $(j - i_{z'})$ single-step skip-transitions.

Example 2. Consider an example where the attractor-based policy synthesis constructs an FSC with ten nodes $n_0, n_1, \ldots, n_9$, where the node n_i was added at the i^{th} iteration. n_0 corresponds to the earliest discovered stationary policy. For each iteration $j > 0$, the newly synthesized subpolicy may need to switch to one of the earlier nodes $\{n_0, \ldots, n_{j-1}\}$ depending on the posterior observation z'. We compare the resulting standard FSC with its skip-FSC variant.

For each node n_j, the transition function $\delta(n_j, z, z')$ must distinguish between up to j different target nodes $\{n_0, \ldots, n_{j-1}\}$, one for each possible winning observation z'. Hence, across all ten nodes, the total number of distinct next-node (in the worst case) labels to encode is $\Sigma_{j=1}^{9} j = 45$. This yields large decision trees with many distinct leaf labels.

In our skip-FSC, we replace each direct jump by a one-step connection to the immediate predecessor: $\gamma_{\text{skip}}(n_j, z) = \text{skip}$ and $\delta_{\text{skip}}(n_j, z, z') = n_{j-1}$, for all winning posterior observations z' and all observation z. When the skip action is played, the controller moves to n_{j-1} and repeats this process until it reaches a node with a non-skip action. Consequently, each n_j has two possible successor (n_j and n_{j-1}) for all winning observations, reducing the number of distinct transition labels to $2 \times 9 = 18$, a $45 \rightarrow 18$ reduction (about $2.5\times$ fewer labels). Thus, the DTs representing transitions become substantially smaller.

We prove that the skip-FSC generated in this way would represent the same policy as the original FSC:

Theorem 1. An FSC $\mathcal{F}$ generated by the iterative approach, and the skip-FSC $\mathcal{F}_{\text{skip}}$ created in Algorithm 3 from it represents the same policy.

Translation from skip-FSC to skip-DT-FSC. Finally, we apply Algorithm 1 to the skip-FSC and represent its components using DTs, resulting in a skip-DT-FSC—essentially a DT-FSC with skip, represented using $\{\text{DT}_{\delta_{\text{skip}},n}\}_{n\in N}$ and $\{\text{DT}_{\gamma_{\text{skip}},n}\}_{n\in N}$.

Remark 3. *For these FSCs, the DT-FSC matches the original policy on all reachable observations. Differences may occur only on unreachable inputs, which do not affect the performance.*

5 EXPERIMENTAL EVALUATION

We adopt the viewpoint that explainability can be proxied by the size of the DTs. Compactness is straightforward to measure using this proxy; however, we further support explainability through a few examples and detailed case studies, demonstrating the interpretability and intelligibility provided by our representation. In particular, we answer the following questions: *Q1): Does our approach make the policies more compact in terms of the size? Q2) Does introducing skip-transitions improves the compactness of policies?*

Compact Representation of Policies. When converting an FSC to a DT-FSC, the number of nodes remains the same. What changes is how the transition function δ and the action mapping γ are represented. For DT-FSCs, instead of representing them in a tabular form, we represent them as a more concise DT. To show the quantitative improvement, we compare the number of rows in the table and the number of nodes in the created DT.

Implementation Details. We consider two sets of benchmarks: (1) For the almost-sure reachability objectives, we use the all the benchmarks from [30]; (2) for the general objectives, we take the benchmarks from [10] which includes models with objectives optimizing quantitative reachability or expected total reward to the target. In the first case, we modified the model checker Storm [29] to get the optimal policies as an FSC. For the general case, we use the implementation in [10]. Our prototype is implemented in Python, which processes the FSCs and uses dtControl [7] with the default settings to create the DTs. *A docker image is publicly available at https://doi.org/10.5281/zenodo.17304441.*

Results. Figure 4 summarizes the results for the benchmark set (1). Our approach achieves significant reduction in size both in the case of **the transition function (by a factor of 13.5 on average) and the action mapping (by a factor of 1.7 times on average)**—answering Q1 (see Table 1 in Section A.4 for details). For the benchmark set (2), the representation size of **the transition function is also reduced by a factor of 5.54 on average and the action mapping is reduced by a factor of 1.66 times on average** (see Table 2 in Section A.4). The action mapping tables are partial and depend only on current observations, while transition functions depend on both current and next observations. This makes action mapping tables smaller in our experiments and explains why DTs save more space in transition functions than in action mappings. For the benchmarks in set (1), **the skip transition reduces the representation size of the transition function by a factor of 16.37 on average and action selection by 1.87**—answering Q2 (see Table 1 in Section A.4).

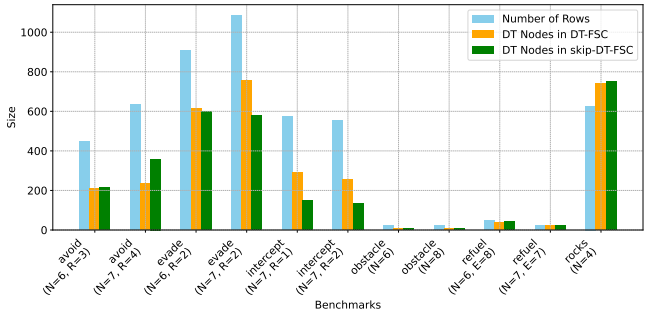
Time. Policy synthesis times vary significantly depending on the benchmark, with larger examples taking over an hour, while DT-FSC translation consistently requires only a few seconds. For instance, across all benchmarks in set(1), policy synthesis took a total of 94 minutes, whereas DT-FSC creation was completed in less than 6 minutes, highlighting the efficiency of DT-FSCs as a lightweight postprocessing step (see Table 3 in Section A.4).

6 EXPLAINABILITY USING CASE STUDIES

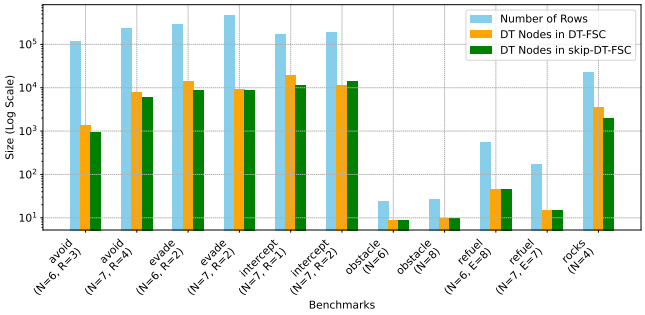
While direct quantitative measurement of explainability is challenging, we aim to assess the explainability of our DT-FSC policies through illustrative case studies. Specifically, we aim to address the question: *Does our proposed approach yield more explainable and practically useful policies?* Below, we detail two case studies to demonstrate the explanatory capabilities of our modular DT-FSC representation. For an additional case study, namely *'Obstacle'* [30] and further details, see Appendix A.3.

6.1 Refuel

We consider a gridworld-based model that was introduced in [22] and modeled later in [30]. A rover must travel from one corner to the diagonally opposite corner in an 6×6 grid while avoiding an obstacle. The rover can observe (i) if the current location is `start` (i.e., at (1,1)), `bad` (i.e., at (5,5)), or `good` (i.e., at (6,6)), (ii) `fuelmeter`:



(a) Different representation sizes of the action mapping γ. *DT-FSC and skip-DT-FSC representations are respectively 1.7 and 1.87 times smaller than the tables.*



(b) Different representation sizes of the transition function δ. *DT-FSC and skip-DT-FSC representations are respectively 13.5 and 16.37 times smaller than the tables.*

Figure 4: Comparison of the sizes of (posterior-aware) FSCs with our (posterior-aware) DT-FSCs for benchmarks obtained from [30]. Log scale is used in Fig. 4b for easier visualization of values spanning different magnitudes.

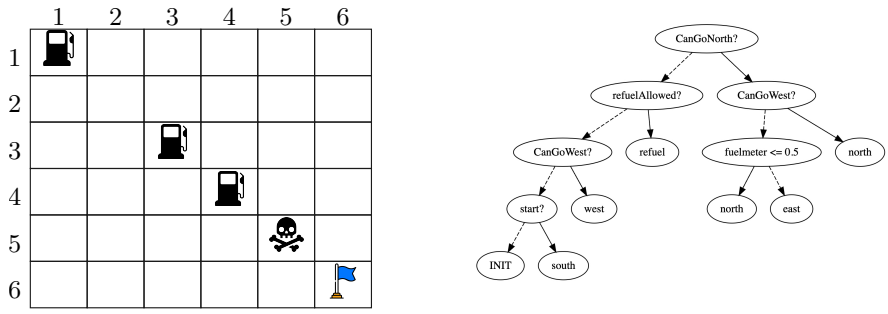


Figure 5: The refuel model for 6×6 grid (Left), the DT representing the policy (Center), and the DT representing the transitions (Right) together illustrate the behavior of the initial node of the DT-FSC. Here, the solid edges in the DT are used to denote `true`, dotted to denote `false`. Initially, the rover cannot move north or west, and by following the decision tree, it first moves south. After this, moving north becomes possible, and the rover then follows the conditions in the DT to move east. If it has not yet reached the refueling station at $(3, 3)$ i.e., refueling is still not possible – it continues moving north until it encounters a wall.*Thus, the initial policy suggests cyclically repeating the sequence of actions (south, east, (north)+, (west)+) where (north)+ and (west)+ corresponds to a sequence of that action until a wall is reached. This allows the agent to get back to its initial state in the corner to refuel or eventually reach $(3, 3)$ safely. It changes to a different node in the FSC once it refuels in $(3, 3)$.

a meter that provides an abstract idea of the current fuel level, (iii) `refuelStation?`: if it is at a refueling station or not (i.e., it is in at a location where it can refuel), (iv) `fuelFull?`: if can be refueled or not, and (v) `CanGo{Direction}` i.e., the cardinal directions in which it can move based on grid-walls. The rover can choose to move one step in any of the directions (unless there is a wall). The grid is slippery, so there is a possibility to slip making it take 2 steps. The rover starts with 8 units of fuel. It consumes one unit of fuel for each step (irrespective of whether it slipped or not) and can fully refuel at the refueling stations located at specific coordinates within the grid. The objective for the rover is to reach a good location while avoiding bad states and preventing fuel depletion.

Explanation of DT-FSC Policies for Refuel. Unlike the tabular representation of FSCs, where the mapping between observations, actions, and transitions appears as large, unstructured lookup tables (with 50 rows describing which action to take and 555 rows describing whether to switch the policy or not), the DT-FSC organizes the same information hierarchically using interpretable predicates. The DTs are significantly smaller (for example, more than 12 times smaller in the case of transition representation). Looking at the DTs, we can thus understand the policy by tracing how each observation leads to an action or memory transition by traversing the DT, revealing the controller's decision logic step by step. We explain the policy in Fig. 5 (Center) and transitions in Fig. 5 (Right) for the initial node of DT-FSC and sketch the remaining policy. The policy suggests cyclically repeating the sequence of actions (south, east, (north)+, (west)+) in order where (north)+ and (west)+ corresponds to a sequence of that action until a wall is reached. with the rover refueling by playing `refuel` whenever possible. This way, the rover will eventually reach the second refueling station at $(3, 3)$, a subgoal in the iterative procedure. Once refueled at the second refuel station, the memory location switches to the next one (see the DT in Fig. 5 (Right)), directing the rover to move south. After reaching the southernmost cell, the policy then suggests the

rover continue east, ultimately reaching the target while avoiding the bad state.

6.2 Planning Treatment of Heart Diseases

We model the diagnosis and treatment of ischemic heart disease as described in [27]. The doctor diagnoses and treats patients potentially suffering from coronary artery disease (CAD) or myocardial ischemia (MI). The model accounts for cases of harmless chest pain, which do not require treatment and the risks of complications arising from diagnostic tests and treatment procedures. The diseases (CAD and MI) are represented by hidden state variables, while the symptoms (e.g., chest pain) are observable. The doctor can use several diagnostic tools with different detection rates: an electrocardiogram (EKG) test, stress test and angiogram. The test results are observable and reveal more information about the hidden state variables and can choose between non-invasive medication, or invasive procedures like coronary artery bypass grafting (CABG) and percutaneous transluminal coronary angioplasty (PCTA). CABG is generally more effective but riskier, while PCTA is less invasive, but its success depends on the presence of complications.

Waiting may resolve harmless symptoms but risks worsening serious conditions. Conducting tests, treatments, and waiting incur some costs. These costs are higher if treatments are administered without proper diagnosis. Additional costs are incurred if complications arise. We modeled this POMDP in the Prism language [35] and used the FSC learning technique with the default configuration implemented in [10] to create a 4 node FSC.

Explanation of DT-FSC Policies for heart diseases. We provide only a partial overview of the policy here. Initially, as described in Fig. 7 in Appendix A.3, the policy suggests `EKGTest`, and depending on the results, advises either additional tests or medication. It suggests going for an invasive procedure only when the stress test is positive. In the later iterations (refer to Section A.3), it suggests randomized policies in the cutoff points. While these policies can be challenging to interpret, they offer some insight into possible actions a doctor might take. For instance, if the policy assigns a higher probability to choosing `pcta` over `medicine`, then PCTA would be preferred in that scenario. Therefore, for complex models like this, a hybrid approach combining DT-FSC with human expertise can offer the needed explainability and an effective policy.

7 CONCLUSION

Our method profits from the graphical representation of FSCs and combines it with the DT representation of (i) the action selection in the memoryless parts of the policy and (ii) the labeling of the transitions in the FSC. Together this provides a modular and flexible framework, able to accommodate, *e.g.*, richer transition functions (taking more observations as arguments) or richer actions (of randomized policies). Not only does it allow for better explainability, but also for better diagnostics of where the complexities arise, *e.g.*, in which situations the posterior observation is really needed or under which constraints randomization occurs (and thus a validation is possibly in place). This could become even more efficient when the modeling process provides the observation and state information in a more structured way, *e.g.*, as values of several sensors rather than an identification number.

ACKNOWLEDGMENTS

This work has been supported by the DFG project GOPro (427755713), MUNI Award in Science Humanities grant (MUNI/I/1757/2021), the ERC project InOVationCS (101171844) and the RSS Scheme (NRF-RSS2022-009) by NRF, Singapore.

REFERENCES

[1] 2021. Proposal for a Regulation laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain Union legislative acts, COM(2021) 206 final - European Commission. https://ec.europa.eu/transparency/regdoc/rep/1/2021/EN/COM-2021-206-F1-EN-MAIN-PART-1.PDF.

[2] Roman Andriushchenko, Alexander Bork, Milan Ceska, Sebastian Junges, Joost-Pieter Katoen, and Filip Macák. 2023. Search and Explore: Symbiotic Policy Synthesis in POMDPs. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France (Lncs)*. Springer. doi:10.1007/978-3-031-37709-9_6

[3] Roman Andriushchenko, Milan Ceska, Sebastian Junges, and Joost-Pieter Katoen. 2022. Inductive synthesis of finite-state controllers for POMDPs. In *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands (Proceedings of Machine Learning Research)*. PMLR. https://proceedings.mlr.press/v180/andriushchenko22a.html

[4] Roman Andriushchenko, Milan Ceska, Sebastian Junges, Joost-Pieter Katoen, and Simon Stupinský. 2021. PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021 (Lecture Notes in Computer Science)*. Springer. doi:10.1007/978-3-030-81685-8_40

[5] Roman Andriushchenko, Milan Češka, Sebastian Junges, and Filip Macák. 2025. Small Decision Trees for MDPs with Deductive Synthesis. *arXiv preprint arXiv:2501.10126* (2025).

[6] Pranav Ashok, Mathias Jackermeier, Pushpak Jagtap, Jan Kretínský, Maximilian Weininger, and Majid Zamani. 2020. dtControl: decision tree learning algorithms for controller representation. In *HSCC*. ACM, 17:1–17:7.

[7] Pranav Ashok, Mathias Jackermeier, Jan Křetínský, Christoph Weinhuber, Maximilian Weininger, and Mayank Yadav. 2021. dtControl 2.0: Explainable Strategy Representation via Decision Tree Learning Steered by Experts. In *TACAS (2) (Lecture Notes in Computer Science, Vol. 12652)*. Springer, 326–345.

[8] Karl Johan Åström. 1965. Optimal control of Markov processes with incomplete state information I. *Journal of mathematical analysis and applications* 10 (1965), 174–205.

[9] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press.

[10] Alexander Bork, Debraj Chakraborty, Kush Grover, Jan Křetínskỳ, and Stefanie Mohr. 2024. Learning Explainable and Better Performing Representations of POMDP Strategies. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

[11] Alexander Bork, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann. 2020. Verification of Indefinite-Horizon POMDPs. In *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12302)*. Springer. doi:10.1007/978-3-030-59152-6_16

[12] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. 1995. Exploiting Structure in Policy Construction. In *IJCAI*. Morgan Kaufmann, 1104–1113.

[13] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Andreas Fellner, and Jan Křetínský. 2015. Counterexample Explanation by Learning Small Strategies in Markov Decision Processes. In *CAV (1) (Lecture Notes in Computer Science, Vol. 9206)*. Springer, 158–177.

[14] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Andreas Fellner, and Jan Křetínský. 2015. Counterexample Explanation by Learning Small Strategies in Markov Decision Processes. In *CAV (1) (Lecture Notes in Computer Science, Vol. 9206)*. Springer, 158–177.

[15] Leo Breiman. 2017. *Classification and regression trees*. Routledge.

[16] Carlos E Budde, Pedro R D'Argenio, and Arnd Hartmanns. 2024. Digging for decision trees: a case study in strategy sampling and learning. In *International Conference on Bridging the Gap between AI and Reality*. Springer, 354–378.

[17] G. Ann Campbell. 2018. Cognitive complexity: an overview and evaluation. In *Proceedings of the 2018 International Conference on Technical Debt* (Gothenburg, Sweden) *(TechDebt '18)*. Association for Computing Machinery, New York, NY, USA, 57–58. doi:10.1145/3194164.3194186

[18] Steven Carr, Nils Jansen, and Ufuk Topcu. 2022. Task-Aware Verifiable RNN-Based Policies for Partially Observable Markov Decision Processes. *J. Artif. Int. Res.* (jan 2022). doi:10.1613/jair.1.12963

[19] Debraj Chakraborty, Clemens Dubslaff, Sudeep Kanav, Jan Kretinsky, and Christoph Weinhuber. 2025. Explaining Control Policies through Predicate Decision Diagrams. *HSCC* (2025).

[20] Krishnendu Chatterjee, Martin Chmelik, and Jessica Davies. 2016. A Symbolic SAT-Based Algorithm for Almost-Sure Reachability with Small Strategies in POMDPs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. AAAI Press, 3225–3232. doi:10.1609/AAAI.V30I1.10422

[21] Krishnendu Chatterjee, Martin Chmelik, Raghav Gupta, and Ayush Kanodia. 2016. Optimal cost almost-sure reachability in POMDPs. *Artificial Intelligence* 234 (2016), 26–48.

[22] Thomas G Dietterich et al. 1998. The MAXQ Method for Hierarchical Reinforcement Learning.. In *ICML*, Vol. 98. 118–126.

[23] Dana Fisman, Hadar Frenkel, and Sandra Zilles. 2023. Inferring symbolic automata. *Logical Methods in Computer Science* 19 (2023).

[24] Amalia Foka and Panos Trahanias. 2007. Real-time hierarchical POMDPs for autonomous robot navigation. *Robotics and Autonomous Systems* 55, 7 (2007), 561–571.

[25] Hector Geffner and Blai Bonet. 1998. Solving large POMDPs using real time dynamic programming. In *Working Notes Fall AAAI Symposium on POMDPs*, Vol. 218.

[26] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* (2019). doi:10.1145/3236009

[27] Milos Hauskrecht and Hamish Fraser. 2000. Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial intelligence in medicine* 18, 3 (2000), 221–244.

[28] Bradley Hayes and Julie A. Shah. 2017. Improving Robot Controller Transparency Through Autonomous Policy Explanation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, HRI 2017, Vienna, Austria, March 6-9, 2017*, Bilge Mutlu, Manfred Tscheligi, Astrid Weiss, and James E. Young (Eds.). ACM. doi:10.1145/2909824.3020233

[29] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. 2022. The probabilistic model checker Storm. *Int. J. Softw. Tools Technol. Transf.* 24, 4 (2022), 589–610. doi:10.1007/s10009-021-00633-z

[30] Sebastian Junges, Nils Jansen, and Sanjit A. Seshia. 2021. Enforcing Almost-Sure Reachability in POMDPs. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12760)*. Springer, 602–625. doi:10.1007/978-3-030-81688-9_28

[31] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1 (1998), 99–134. doi:10.1016/S0004-3702(98)00023-X

[32] Omar Zia Khan, Pascal Poupart, and James P. Black. 2009. Minimal Sufficient Explanations for Factored Markov Decision Processes. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece*, Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis (Eds.). AAAI.

[33] Jonis Kiesbye, Kush Grover, Pranav Ashok, and Jan Křetínskỳ. 2022. Planning via model checking with decision-tree controllers. In *2022 International Conference On Robotics And Automation (ICRA)*. IEEE.

[34] Anurag Koul, Sam Greydanus, and Alan Fern. 2018. Learning Finite State Representations of Recurrent Policy Networks. *ArXiv* abs/1811.12530 (2018). https://api.semanticscholar.org/CorpusID:54434799

[35] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA (Lecture Notes in Computer Science)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer. doi:10.1007/978-3-642-22110-1_47

[36] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. 1995. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*. Elsevier, 362–370.

[37] Omid Madani, Steve Hanks, and Anne Condon. 2003. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence* 147, 1-2 (2003), 5–34.

[38] Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. [n. d.]. Explainable Reinforcement Learning through a Causal Lens. In *The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, New York, NY, USA*. doi:10.1609/AAAI.V34I03.5631

[39] Oded Z Maimon and Lior Rokach. 2014. *Data mining with decision trees: theory and applications*. Vol. 81. World scientific.

[40] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.* 267 (2019). doi:10.1016/J.ARTINT.2018.07.007

[41] Daniel Neider and Oliver Markgraf. 2019. Learning-Based Synthesis of Safety Controllers. In *FMCAD*. IEEE, 120–128.

[42] Judea Pearl. 2009. *Causality: Models, Reasoning and Inference* (2nd ed.). Cambridge University Press, USA.

[43] David Silver and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. *Advances in neural information processing systems* 23 (2010).

[44] Richard D. Smallwood and Edward J. Sondik. 1973. The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Oper. Res.* 21, 5 (1973), 1071–1088. doi:10.1287/opre.21.5.1071

[45] Pulkit Verma and Julie A. Shah. 2025. Interpretability Analysis of Symbolic Representations for Sequential Decision-Making Systems. In *HRI 2025 Workshop on Explainability for Human-Robot Collaboration: Real-World Concerns*.

A APPENDIX

A.1 Additional Details of Section 3

Fig. 6 shows the policy for Example 1. For observations b, y, policies are defined in both memory states, while g and w trigger a switch.

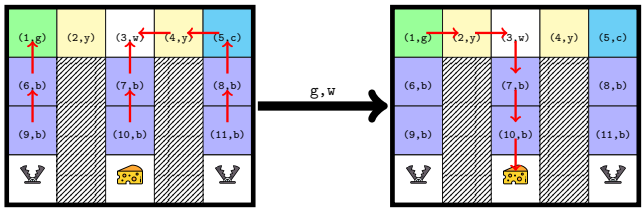


Figure 6: Policy of n_0 (Left) and n_1 (Right) in Maze and the transitions on the bold arrow. The numbers are to indicate different states and the letters are used for observations. Same letter for cells implies that the mouse can not distinguish between those states.

A.2 Proof of Correctness for FSC with Skip

For our new skip-FSC, we first recall the following observation.

Observation 1. *For any winning observation $z \in Z$, there exists an iteration i_z where we find a policy that is winning from any state with observation z. Then in the FSC generated by the iterative approach, for any $j > i_z$, for any $z' \in Z$, $\delta(n_j, z', z) = n_{i_z}$.*

Next we formally prove the correctness of Algorithm 3.

Theorem 1. An FSC $\mathcal{F}$ generated by the iterative approach, and the skip-FSC $\mathcal{F}_{\text{skip}}$ created in Algorithm 3 from it represents the same policy.

Proof. We show that for any observation sequence ρ in Z^*, starting from the initial node n_{init}, both the original $\mathcal{F}$ and the $\mathcal{F}_{\text{skip}}$ will reach the same node and suggest same action. This is proven by induction on $|\rho|$.

Base case : $\rho = O(s_{\text{init}})$. We stop the iteration when $O(s_{\text{init}})$ is in the winning region. So both the FSCs would suggest $\gamma(n_{\text{init}}, O(s_{\text{init}}))$.

Induction step : Assume the claim is true for some observation sequence ρ, meaning we reach the same node n in both $\mathcal{F}$ and $\mathcal{F}_{\text{skip}}$. Now consider an extended observation sequence $\rho \cdot z$. Let $\delta(n, \text{last}(\rho), z) = n'$. There are two cases to consider.

Case 1: $n' = n_{i_z}$ where i_z is the index of the node defined in Observation 1. From Algorithm 3, we have $\gamma_{\text{skip}}(n, z) = \text{skip}$ for all nodes from n to n_{i_z}. Then, for the observation sequence $\rho \cdot z$, we reach the same node n_{i_z} in both $\mathcal{F}$ and $\mathcal{F}_{\text{skip}}$. Also, as we are not adding any skip actions from z at n_{i_z}, we have $\gamma_{\text{skip}}(n_{i_z}, z) = \gamma(n_{i_z}, z)$.

Case 2: $n' \neq n_{i_z}$ This implies that $\delta(n, \text{last}(\rho), z) = \delta_{\text{skip}}(n, \text{last}(\rho), z) = n'$ and $\gamma_{\text{skip}}(n_{i_z}, z) \neq \text{skip}$. So $\gamma_{\text{skip}}(n_{i_z}, z) = \gamma_{\text{skip}}(n_{i_z}, z)$.

Thus, the original FSC $\mathcal{F}$ and the skip-FSC $\mathcal{F}_{\text{skip}}$ both reach the same node and suggest the same action for any observation sequence ρ. Therefore, $\mathcal{F}$ and $\mathcal{F}_{\text{skip}}$ represent the same policy. □

A.3 Additional details on Case Studies

Heart diseases. We show all the policies (Fig. 7 to Fig. 10) and transitions (Fig. 11) for the heart example. Observe that the next policies suggest randomizing between different procedures - which gives a hint based on probabilities what could be a better choice. However, it also shows that we need human intervention in order to take more rational decisions.

Obstacle. This case study involves a robot navigating through an environment with static obstacles to reach a designated exit. The environment is partially observable, adding complexity to the decision-making process. The robot begins in an initial state that is uncertain.

The robot has limited observational capabilities; it can only determine whether its current position is either an "obstacle" (an area that prevents further progress and penalizes the robot) or the "exit" (the target destination). This restricted sensory input mirrors realistic scenarios where robots operate with incomplete or noisy information about their surroundings.

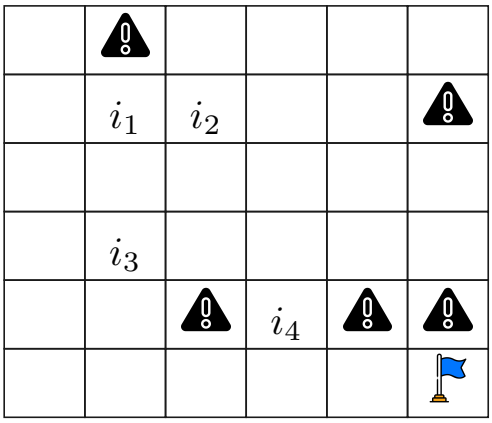


Figure 13: The obstacle case study with 6×6 grid, 5 obstacles, and 4 non-deterministic initial locations (i_1, i_2, i_3, and i_4) of the robot.

Explanation of DT-FSC Policies for Obstacle. The DT-FSC policy generated by our approach demonstrates how the robot can navigate the grid safely, regardless of its starting position (i_1, i_2, i_3, or i_4). The initial stationary policy advises the robot to take a single step towards `south`. Afterward, the policy utilizes three memory nodes to guide the robot through a series of steps: first moving completely to the west playing `west` action 3 times (note the initial position i_4), then taking three steps `south`, and finally continuing to the `east` until it reaches the target. Each memory location in the policy contains only a single decision tree (DT) node, which dictates the direction the robot should move. Essentially, the policy incorporates an implicit step-counting mechanism, as the robot can only observe whether it has reached the target or encountered an obstacle.

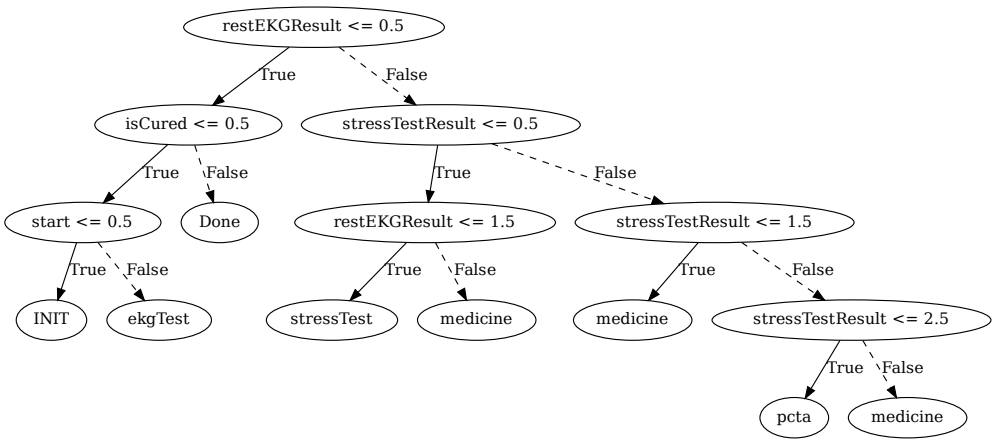


Figure 7: DT representation of the stationary policy at node 0 for treatment of heart diseases

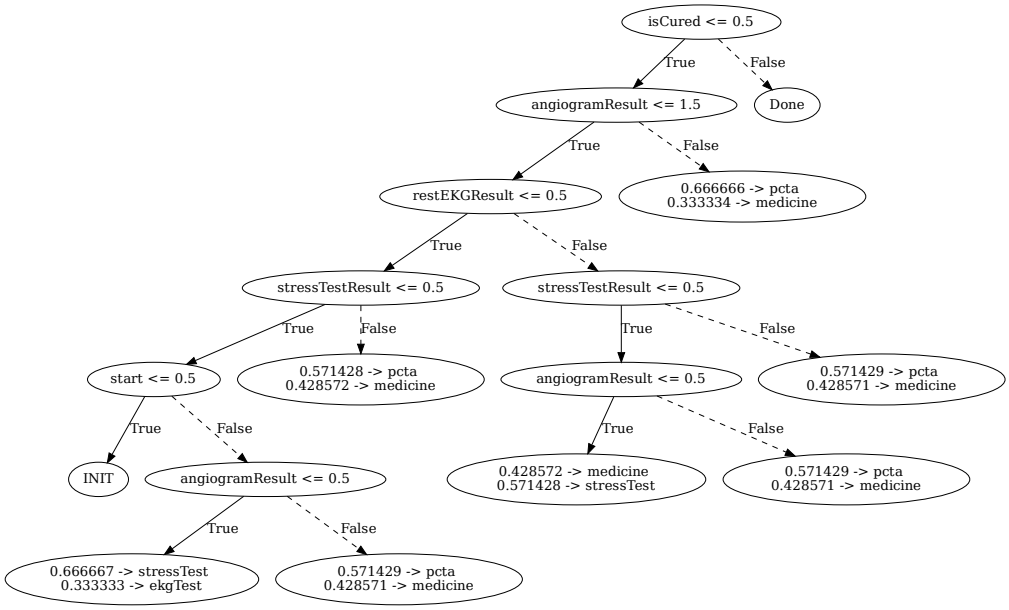


Figure 8: DT representation of the stationary policy at node 3 for treatment of heart diseases

A.4 Additional Details on Experimental Evaluation

We present a summary of additional experimental results here. Additionally, we provide the data to generate these values. The instruction are in the `README.md` accompanied by this Appendix. Moreover, we provide a link to a publicly available docker image which can be used to reproduce the results from scratch.

Effect of skip Actions. Table 1 demonstrates that when comparing the number of rows in a FSC in tabular form to a DT-FSC created from the skip-FSCs, the latter is more efficient. Specifically, the tables have 1.87 times more rows as compared to the number of nodes in DT representation of the policies in the skip-DT-FSC and 16.37 times more rows as compared to the number of nodes in DT representation of the transitions in the skip-DT-FSC.

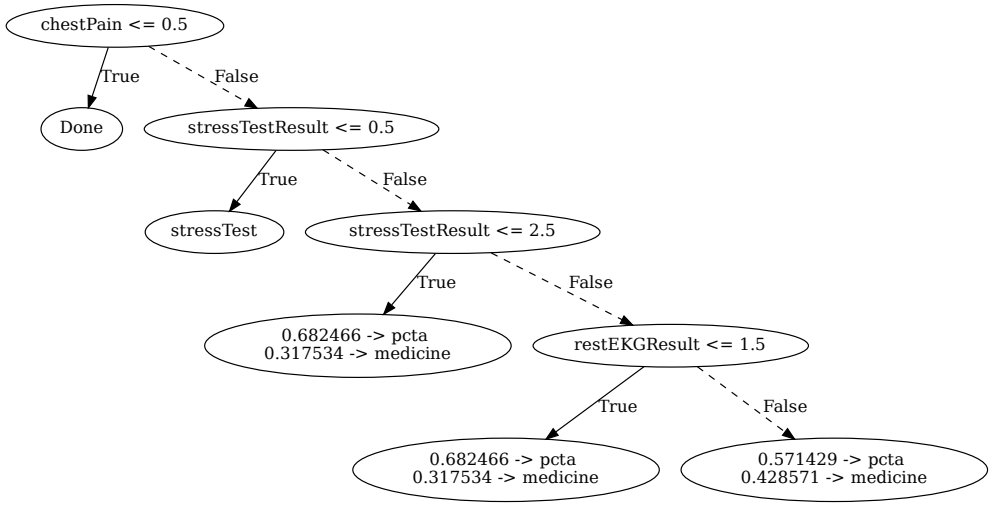


Figure 9: DT representation of the stationary policy at node 9 for treatment of heart diseases

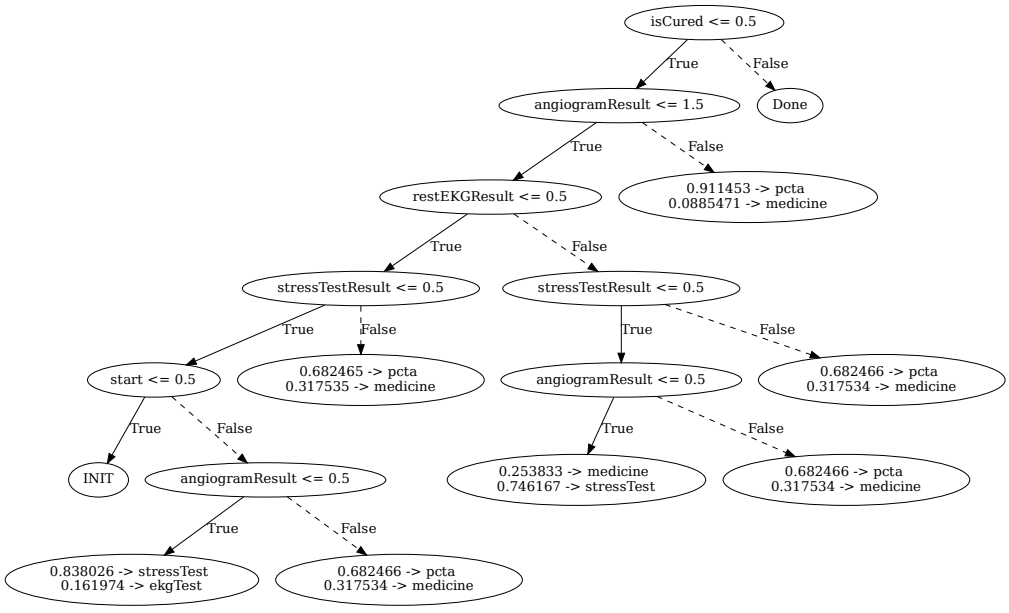
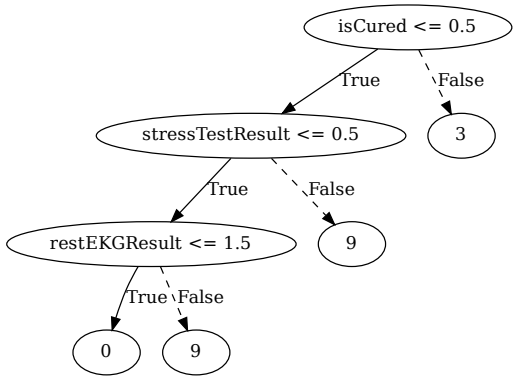


Figure 10: DT representation of the stationary policy at node 11 for treatment of heart diseases

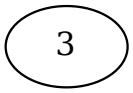
Results for Benchmark Set (1). For benchmark set (1), we compare the sizes of (posterior-aware) FSCs with our (posterior-aware) DT-FSCs for benchmarks obtained from [30]. Table 1 describes the results. Table 2 describes the results.

Results for Benchmark Set (2). For benchmark set (2), we generate DT-FSCs from the FSCs created in [10] for quantitative benchmarks, we achieve a significant reduction in transition sizes. Table 2 describes the results.

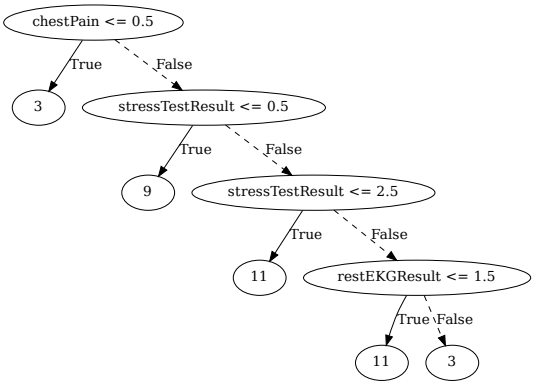
Time. Table 3 reports the time taken for synthesizing the policy and creating the DT-FSCs from them. While the model checker Storm can leverage parallelization, learning the DTs were done sequentially. Even then, creating DT-FSCs from synthesized policies takes less than 2 minutes per benchmark, even for large benchmarks,



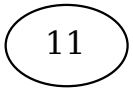
(a) From node 0



(b) From node 3



(c) From node 9



(d) From node 11

Figure 11: DT representation of the FSC transitions from different nodes in the FSC for treatment of heart diseases

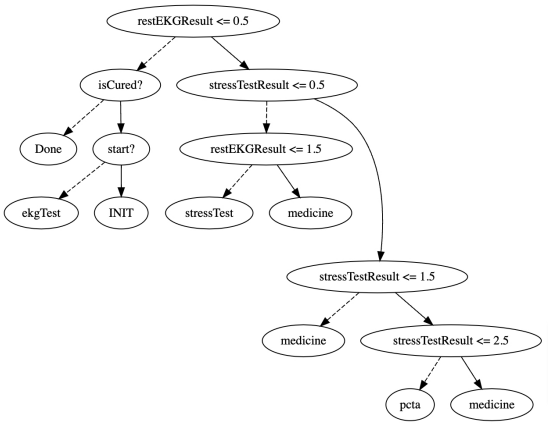


Figure 12: Decision tree representation of the initial heart disease treatment policy from the FSC. Test results: 0 = not done, 1 = negative, 2 = positive, 3 = inconclusive.

while policy synthesis can take over an hour for a large benchmark (see Table 5 in the Appendix for a detailed time comparison). In total, **for the benchmarks in Set 1, the policy synthesis took 94 minutes, whereas DT-FSC creation was completed in less than 6 minutes**. This highlights the efficiency of using DT-FSCs as a lightweight postprocessing step that adds explainability with minimal time overhead.

Table 1: Comparison of the sizes of (posterior-aware) FSCs with our (posterior-aware) DT-FSCs and skip -DT-FSC for benchmark set (1) obtained from [30].

Benchmark	#Nodes	Stationary Policy Size					Transition Size				
		#Rows	#DT-nodes	Ratio (FSC/ DT-FSC)	#Skip-DT-nodes	Ratio (FSC/ Skip-DT)	#Rows	#DT-nodes	Ratio (FSC/ DT-FSC)	#Skip-DT-nodes	Ratio (FSC/ Skip-DT)
avoid (N=6, R=3)	9	447	209	2.14	215	2.08	121837	1381	88.21	945	129.01
avoid (N=7, R=4)	3	634	233	2.72	359	1.77	240418	7927	30.32	6135	39.19
evade (N=6, R=2)	31	908	613	1.48	599	1.52	296100	14003	21.14	8607	34.39
evade (N=7, R=2)	30	1085	758	1.43	578	1.88	477329	9226	51.73	8818	54.15
intercept (N=7, R=1)	5	575	289	1.99	149	3.86	176714	19551	9.04	11549	15.30
intercept (N=7, R=2)	5	554	255	2.17	133	4.17	189017	11421	16.55	14157	13.35
obstacle (N=6)	7	22	9	2.44	9	2.44	24	9	2.67	9	2.67
obstacle (N=8)	8	25	10	2.50	10	2.50	27	10	2.70	10	2.70
refuel (N=6, E=8)	5	50	39	1.28	45	1.11	555	45	12.33	45	12.33
refuel (N=7, E=7)	3	24	23	1.04	23	1.04	172	15	11.47	15	11.47
rocks (N=4)	52	627	740	0.85	750	0.84	22634	3578	6.33	1968	11.50
Geometric Mean				**1.7**		**1.87**			**13.5**		**16.37**

Table 2: Comparison of FSC sizes with our DT-FSC sizes for the benchmark set (2) obtained from [10]

Benchmark	#FSC-nodes	Stationary Policy size			Transition size		
		#Rows	#Nodes	Ratio (Rows/Nodes)	#Rows	#Nodes	Ratio (Rows/Nodes)
refuel-06	3	71	75	0.95	89	19	4.68
grid-large-30-5	1	39	39	1.00	40	1	40.00
problem-storm-extended	61	246	65	3.78	247	63	3.92
grid-avoid-4-0	5	24	13	1.85	26	9	2.89
posterior-awareness	4	20	8	2.50	20	12	1.67
grid-avoid-4-0	5	22	9	2.44	24	7	3.43
hallway	18	198	206	0.96	198	144	1.38
4x3-95	9	71	63	1.13	71	55	1.29
stand-tiger-95	4	28	18	1.56	28	14	2.00
query-s3	16	104	72	1.44	104	66	1.58
grid-large-20-5	1	19	19	1.00	20	1	20.00
problem-storm	3	14	7	2.00	15	5	3.00
rocks-16	2	2621	826	3.17	2774	4	693.50
query-s2	9	62	39	1.59	62	37	1.68
network-prio-2-8-20	1	4896	429	11.41	4912	1	4912.00
problem-storm-paynt-combined	7	40	29	1.38	41	21	1.95
network	5	29	23	1.26	29	11	2.64
milos-aaai97	3	24	13	1.85	24	5	4.80
refuel-08	5	113	115	0.98	136	43	3.16
hallway2	16	317	262	1.21	317	206	1.54
rocks-12	2	1541	542	2.84	1658	4	414.50
web-mall	3	17	11	1.55	17	7	2.43
4x5x2-95	18	107	86	1.24	107	74	1.45
problem-paynt-storm-combined	6	39	26	1.50	42	14	3.00
maze-alex	16	119	114	1.04	122	84	1.45
grid-large-10-5	8	36	14	2.57	37	8	4.63
problem-paynt	3	13	5	2.60	13	5	2.60
mini-hall2	3	25	25	1.00	25	15	1.67
refuel-20	4	194	196	0.99	227	22	10.32
Geometric Mean				**1.66**			**5.54**

Table 3: Comparison of time (in seconds) taken for synthesizing the policies and the time taken for creating the DT-FSCs as a postprocess step for the benchmarks obtained from [30].

Benchmarks	#States	#Transitions	#Observations	Synthesis Time (sec.)	DT-FSC Creating Time (sec.)
avoid (N=6, R=3)	5976	14373	3300	57.65	20.56
avoid (N=7, R=4)	13021	33949	8584	4715.34	23.31
evade (N=6, R=2)	4232	28866	2202	144.97	70.18
evade (N=7, R=2)	8108	57570	4172	541.93	77.04
intercept (N=7, R=1)	4705	18049	2002	60.72	22.20
intercept (N=7, R=2)	4705	18049	2598	55.24	20.99
obstacle (N=6)	37	224	4	0.17	10.51
obstacle (N=8)	65	421	4	0.31	12.06
refuel (N=6, E=8)	270	1301	36	1.27	7.52
refuel (N=7, E=7)	302	1545	35	0.97	4.55
rocks (N=4)	331	3484	65	64.66	82.89
Total time				5643.25	351.81

B REPRODUCIBILITY

To ensure the reproducibility of our results, a docker image is publicly available at the following link:

https://doi.org/10.5281/zenodo.17304440

B.1 Experimental Environment

The experiments were conducted on a host machine with the following specifications:

CPU: AMD Ryzen 7 PRO 6850U (16 cores, 2.7 GHz base clock)
RAM: 32 GB
Operating System: Ubuntu 24.04.1 LTS with Linux kernel 6.8.0-51

All experiments were executed within a Docker container using Docker version 26.1.3. The container was not restricted by any explicit resource limits, allowing it to fully utilize the resources of the host machine.

B.2 Docker Artifact

The following commands demonstrate how to use the Docker artifact:

```
# Load the Docker image
docker load < pomdp-dt-fsc.tar

# Run the Docker container interactively
docker run -it pomdp-dt-fsc:latest /bin/bash

# Navigate to the repository
cd pomdp-explainable-policy/
```

The docker includes scripts to reproduce both benchmark set (1) (qualitative anaysis) and benchmark set (2) (quantitative analysis):

Qualitative Analysis: Navigate to the respective directory and run:

```
./run_all.sh
```

Quantitative Analysis: Navigate to the quantitative directory and execute:

```
./run_quantitative.sh
```