Modeling Blackbox Agent Behaviour via Knowledge Compilation

Christian Muise[1], Salomón Wollenstein-Betech[2], Serena Booth[3], Julie Shah[3], and Yasaman Khazaeni[4]

[1] School of Computing, Queen's University
[2] Division of Systems Engineering, Boston University
[3] CSAIL, Massachusetts Institute of Technology
[4] IBM Research AI

muise@cs.queensu.ca salomonw@bu.edu sbooth@mit.edu
julie_a_shah@csail.mit.edu yasaman.khazaeni@us.ibm.com

Abstract. Understanding how agents behave in an environment is a cornerstone of interpretable AI. In this work, we focus on capturing and summarizing the policy an agent is following *without* placing any assumptions on how that policy is actually implemented. From a corpus of state-action pairs, we build a compact and diagnosable representation of the mapping from states to actions. We appeal to modern knowledge compilation techniques for this task and demonstrate empirically how this approach outperforms the previous state of the art. We further create an interactive interface to allow people to explore the compiled representation and develop their mental models of the policy. Interface, implementation, and evaluation will be released in full upon publication.

Keywords: behaviour modeling · knowledge compilation · d-DNNF

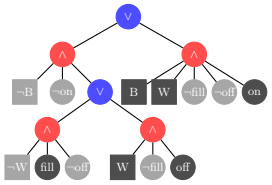
1 Introduction

Understanding an agent's behaviour is a key challenge in Artificial Intelligence. Successfully doing so could have far-reaching implications in multi-agent systems, business process mining, and general dynamical system diagnosis. The introduction of the European regulation GDPR bolstered demand for explanation of algorithmic decision making and paved the pathway toward explicit "right to an explanation" laws [6]. Our work contributes to explainable AI and directly addresses the problem of understanding an agent's policy.

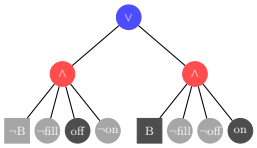
Broadly speaking, our task is to succinctly capture the mapping of states to actions that the agent is using given only historical data: not only can this provide gains in efficiency for exploring the policy, but it can also elucidate key properties of the agent's policy that are only evident by observing their behaviour holistically. Crucially, we make no assumption on the precise mechanics of the agent's policy implementation: it could be a simple program, a neural network approach that is mapping states to actions, or even a human agent with uncertain policy modeling. The agent in this setting is a black box.

Boiling Water	**W**ater in Pot	Action
True	True	turn_**on**_stove
False	True	turn_**off**_stove
False	False	**fill**_pot

(a) Data



(b) Compiled d-DNNF



(c) Conditioned on W

Fig. 1: Example policy compilation. *B*, *W*, *On*, *Off*, *Fill* respectively correspond to `Boiling_Water`, `Water_In_Pot` `Turn_On_Stove`, `Turn_Off_Stove`, and `Fill_in_pot`.

In this work, we assume a shared syntax for the acting agent and the observed behaviours: the fluents in the state of the world and in the actions executed are known and common to both agent and observer.

Our framework closely follows the work of Hayes and Shah [7]. The key distinction of our work is the computational core we employ for compiling the policy. Hayes and Shah's approach relies on the Quine-Mcluskey minimization method for logical inference, which results in an exponential computational blow-up. We instead appeal to state-of-the-art knowledge compilation methods and use disjunctive decomposable negation normal form (d-DNNF) as the key representation [5]. We found that d-DNNF provides a necessary mix of compactness, computationally efficient compilation, and expressive inference capabilities.

We view the historical data of state-action tuples as clauses in a particular logical theory, e.g., disjunctive normal form (DNF). We then compile this theory to various forms of conjunctive normal form (CNF). From the CNF representation, we then use off-the-shelf compilers to produce a d-DNNF representation to capture salient properties of the policy. For example, consider the observation list in Figure 1a and the corresponding d-DNNF representation in Figure 1b.

An added benefit of using d-DNNF as the core representation is that a wide variety of subsequent inference questions remain polynomial in the size of the representation. This allows us to pay the initial cost of compiling the policy only once, and then interactively condition the policy to answer questions such as "*When would the agent perform action X?*" or "*What would the agent do if Y was true and Z false?*". We captured this functionality in an interface equipped to handle such human-in-the-loop policy exploration.

Knowledge compilation technology has a long history of practical advances that make it a competitive solution to settings of logic-based inference. We demonstrate the improvement this lends our approach by comparing with the previous work in this specific setting of agent behaviour modeling, tested on a variety of domains from planning and RL. We also provide an empirical comparison among the different CNF varieties that we investigated in the context of this work, pointing to the preferred method of compilation.

We begin by detailing some of the preliminary concepts we employ. Next, we present the core encodings to CNF that we consider in Section 3 and follow with an empirical evaluation in Section 4. We conclude with a look at related literature and a summary including future directions in Sections 5 and 6.

2 Preliminaries

2.1 Policy Learning

Despite the recent success of decision-making agents, these agents are usually complex and uninterpretable due to their requisite handling of high dimensional inputs and outputs. Gaining insight into the behaviour of such systems is necessary to provide explanation to users (e.g., for prescription recommenders), to facilitate debugging, and to unveil the strengths and weaknesses of agents. Our goal is to reason over an agent's decisions and answer questions about its underlying logic independent of its internal processes. We restrict our work to discrete domains of both states and actions. Of course, one might use techniques of discretization to parse continuous to discrete domains.

We assume we observe the state as fluents $f \in \mathcal{F}$. We denote the state-space (environment) as $\mathcal{S} \subseteq 2^{\mathcal{F}}$ and a particular state of the world to be $s \in \mathcal{S}$. We use $\mathcal{A}$ to represent the action space and $a \in \mathcal{A}$ to specify an action. Let a state-action observation be a tuple $\langle s, a \rangle$ and our data be $\mathcal{D} = \{\langle s_1, a_1 \rangle, ..., \langle s_m, a_m \rangle\}$. The problem we address is to *succinctly represent the mapping* $\mathcal{P} : \mathcal{S} \rightarrow \mathcal{A}$.

The *curse of dimensionality* prohibits the naive approach of obtaining a mapping from states to actions in a big lookup table of the state-action pairs. Some efforts to solve this problem use Quine-Mcluskey minimization [7], Bayesian probabilistic models [9], among others. See more related work in section 5. We propose using Knowledge Compilation to overcome this computational burden, and explore a variety of possible encodings to accomplish this task.

2.2 Knowledge Compilation

The objective of Knowledge Compilation is to build a structured representation of logical theories to allow for subsequent tractable operations. The idea lies primarily on compiling a propositional theory off-line into a target language, which then is used on-line to perform fast operations and reasoning. In the context of our work, we use deterministic decomposable negation normal form (d-DNNF) as our target language. This language allows polynomial time operations (on a pre-compiled theory) such as *conditioning* and *model counting* which are the core for our inference and interaction tool. Although the off-line compilation process may be computationally expensive, it is just performed once.

Negation Normal Form (NNF) Many languages in Knowledge Compilation are subsets of NNF [5]. In NNF, the only allowed Boolean operators are conjunction ($\wedge$, `and`) and disjunction ($\vee$, `or`). The negation operator ($\neg$, `not`) is only applied to variables. A common and practical representation of NNF languages is to construct a directed acyclic graph (DAG). In this graph, inner nodes are either conjunction or disjunction and the leaves are positive or negative variables. Denote Σ as a propositional theory (a DAG). Let C be any node in Σ and $\texttt{Vars}(C)$ the set of variables appearing in the subgraph rooted at C.

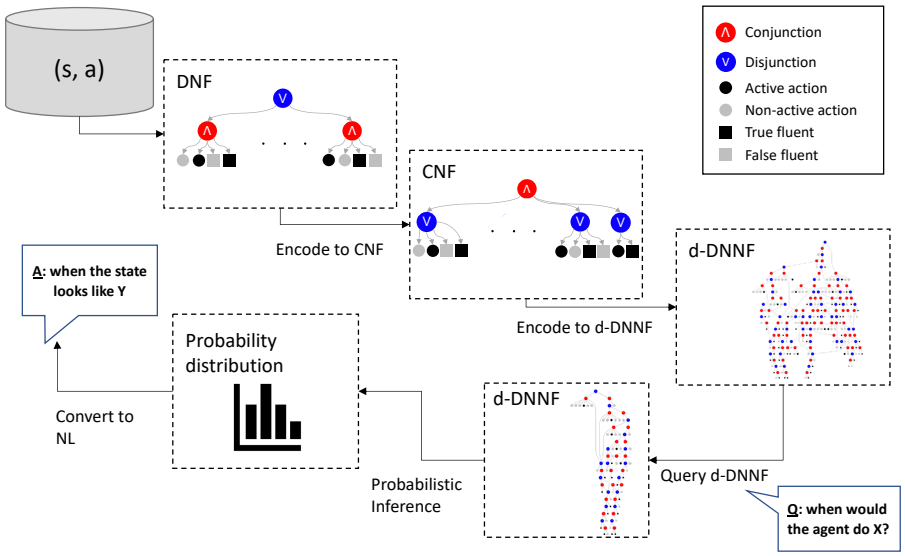


Fig. 2: Overall approach for compiling agent behaviour into a form for inspection.

Disjunctive Normal Form (DNF) This language consists of a disjunction of conjunctions (`or` of `and`'s). Its DAG is *flat*, meaning that the distance from root to any leaf is 2. We use this language to represent our data $\mathcal{D}$. For each state-action pair $\langle s_i, a_i \rangle$ we construct a clause C_i, then take the disjunction over all of the C_i's. See first step in Figure 2.

Conjunctive Normal Form (CNF) In similar fashion as above, CNF is a conjunction of disjunctions (`and` of `or`'s) whose DAG is also *flat* (Figure 2). The purpose of using this intermediate language between our data (DNF) and target (d-DNNF) is twofold. (1) To use efficient off-the-shelf CNF→d-DNNF compilers such as DSHARP [13], c2d [4] or D4 [10], and (2) to allow for encoding with the different properties (cf. Table 1) which we elaborate in section 3.

Deterministic Decomposable Negation Normal Form (d-DNNF) This language requires two additional characteristics over a general NNF: *decomposability* and *determinism*. The objective of imposing such properties is to allow for fast on-line inference. In particular, the ability to perform the operations of *model counting* and *conditioning* in polynomial time over the propositional theory. We now present the definitions [5] of such properties.

Definition 1 (Decomposability). *An NNF satisfies the decomposability property if for any conjunction C, the conjuncts of C do not share any variable. If $C_1, ..., C_n$ are children of an* `and` *node C, then* $\texttt{Vars}(C_i) \cap \texttt{Vars}(C_j) = \emptyset$ *for $i \neq j$.*

Definition 2 (Determinism). *An NNF satisfies the determinism property if for any disjunction C, every pair of disjuncts of C are logically contradictory. That is, if $C_1, ..., C_n$ are children of an* `or` *node C, then $\forall i, j \in [1, ..., n]$ where $i \neq j$, $C_i \wedge C_j =$* `False`*.*

The logic structure of d-DNNF enables counting the number of models in a propositional theory in polynomial time. Without loss of generality, it is possible to count the number of models of a d-DNNF Σ by replacing the `or` nodes by *additions*, the `and` nodes by *products*, and leaf nodes by 1. The NNF must also be *smooth* to obtain a normalized count. Refer to [5] for further details.

Definition 3 (Conditioning). *Let Σ be a deterministic decomposable NNF (d-DNNF). Conditioning on variable x, i.e. $\Sigma|x$, results in the DAG computed by replacing x by* `True` *and $\neg x$ by* `False` *and propagating this information throughout the DAG using standard logical rules. For example, if $\Sigma = (x \vee y) \wedge (\neg x \vee z)$, then, $\Sigma|x = (\texttt{True} \vee y) \wedge (\texttt{False} \vee z)$ which simplifies to z.*

Through the combination of conditioning and counting, the likelihood of a particular variable can be computed as:

$$\Pr(x = \texttt{True}) = \frac{\texttt{count}(\Sigma|x)}{\texttt{count}(\Sigma)}$$

3 Approach

The overall approach is shown in Figure 2. The initial step defines a CNF encoding for the historical data of state-action pairs. We consider a variety of possible encodings for this, and explore the ramifications of each.

The remaining steps use a compiler to convert from CNF to d-DNNF; optionally apply conditional inference on the resulting compilation; and then return both the resulting theory and probability distributions as part of a targeted response. For the compilation, we found Dsharp [13] to be the most efficient option for these encodings (with comparisons made to c2d [4] and D4 [10]).

We begin by looking at the core encoding components considered in 3.1, followed by the properties of the resulting theory in 3.2, and conclude this section with a discussion of the inference and interaction components in 3.3.

3.1 Encoding

Common to all of the encodings is the set of boolean variables that are used to represent the historical data (some encodings will introduce further auxiliary variables, and are discussed later). We begin by defining the two core variable types before expanding on the variety of encoding options.

Definition 4 (Fluent and Action Variables). *For every fluent $f \in \mathcal{F}$, we associate a boolean variable x_f to represent the notion that f was true in a state. For every action $a \in \mathcal{A}$, we associate a boolean variable x_a to represent the notion that the agent performed action a.*

We define four encoding options that can be mixed and matched to produce a theory with a variety of properties. We begin by providing one complete example, and then elaborating on the four orthogonal variations that can be deployed.

Definition 5 (Base Encoding). *Assume $\mathcal{D}$ is a set of tuples of the form $\langle s, a\rangle$ where $s \subseteq \mathcal{F}$ and $a \in \mathcal{A}$. The* Base Encoding *is defined as:*

$$\bigvee_{\langle s,a\rangle \in \mathcal{D}} \left(x_a \wedge \bigwedge_{a' \in \mathcal{A}\setminus\{a\}} \neg x_{a'} \wedge \bigwedge_{f \in s} x_f \wedge \bigwedge_{f \notin s} \neg x_f \right) \tag{1}$$

There are four key parts to each term of the base encoding: (1) the action a must be the one we select; (2) no other action is selected; (3) all of the true fluents in s must be true in the encoding; and (4) all of the false fluents (i.e., those not in s) must be false in the encoding. The target language for knowledge compilation is CNF, however the theory in Definition 5 is in Disjunctive Normal Form (DNF). To remedy this, we apply the well-known Tseitin encoding [17]. Doing so introduces a new auxiliary variable, x_t, for each term t in the DNF above, and the CNF contains clauses of the following form for each term t:

$$\textit{Assuming } \; t = (x_1 \wedge \cdots \wedge x_n)$$

$$\begin{aligned} x_t &\rightarrow (x_1 \wedge \cdots \wedge x_n) \\ \neg x_t &\vee (x_1 \wedge \cdots \wedge x_n) \\ (\neg x_t &\vee x_1) \wedge \cdots \wedge (\neg x_t \vee x_n) \end{aligned}$$

$$\begin{aligned} (x_1 \wedge \cdots \wedge x_n) &\rightarrow x_t \\ \neg(x_1 \wedge \cdots \wedge x_n) &\vee x_t \\ (\neg x_1 \vee \cdots \vee \neg x_n &\vee x_t) \end{aligned}$$

One final clause is included to indicate that at least one of the conjunctive terms $(t_1 \cdots t_m)$ is true: $(x_{t_1} \vee \cdots \vee x_{t_m})$ We examine this encoding's properties later in Section 3.2, but now turn our attention to the variations of the theory.

Onehot (OH) The base encoding ensured two things for each tuple in the data $\mathcal{D}$ with respect to the actions: (1) the appropriate action executes and (2) no other action does. We refer to this combination of terms as the *onehot* variant. When disabled, we only adopt point (1), and forgo mentioning the other actions which were not executed. This fundamentally changes the theory, but,

in combination with other encoding varieties, may be useful in practice. By disabling the onehot property of the base encoding, we have:

$$\bigvee_{\langle s,a\rangle\in\mathcal{D}} \left(x_a \wedge \bigwedge_{f\in s} x_f \wedge \bigwedge_{f\notin s} \neg x_f \right) \tag{2}$$

Implication (IMP) The relationship in the base encoding between the state and the action is via a conjunction. If instead we viewed the theory as a set (conjunction) of implications between the state and the actions that were executed in them, then we arrive at the following theory (by using the *implication* encoding for Formula (2)):

$$\begin{aligned} &\bigwedge_{\langle s,a\rangle\in\mathcal{D}} \left(\left(\bigwedge_{f\in s} x_f \wedge \bigwedge_{f\notin s} \neg x_f \right) \rightarrow x_a \right) \\ =\ &\bigwedge_{\langle s,a\rangle\in\mathcal{D}} \left(\neg \left(\bigwedge_{f\in s} x_f \wedge \bigwedge_{f\notin s} \neg x_f \right) \vee x_a \right) \\ =\ &\bigwedge_{\langle s,a\rangle\in\mathcal{D}} \left(\bigvee_{f\in s} \neg x_f \vee \bigvee_{f\notin s} x_f \vee x_a \right) \end{aligned} \tag{3}$$

At Most One Action (AMOA) As an option that can be combined with any of the previous encodings via a simple conjunction, we consider enforcing the notion that *at most one action* can be executed. This is achieved by adding a binary clause for every pair of unique actions to the theory:

$$\bigwedge_{a\in\mathcal{A}} \bigwedge_{a'\in\mathcal{A}\setminus\{a\}} (\neg x_a \vee \neg x_{a'}) \tag{4}$$

Since it is already CNF, combination with other theories is simply a conjunction.

At Least One Action (ALOA) Similar to AMOA, we use a single clause enforcing *at least one action* be executed:

$$\bigvee_{a\in\mathcal{A}} x_a \tag{5}$$

With the conjunction of (4)+(5), the theory ensures *exactly* one action is true.

3.2 Properties

With four encoding varieties, there are in fact 2^4 possible encodings to consider: every combination of settings yields a viable encoding with different properties. Here, we identify the three key properties that dictate the space of possible boolean functions, and describe how the encoding varieties lead to the combination of properties.

CNF Encoding				Properties		
AMOA	ALOA	OH	IMP	All states covered	Deteministic behaviour	Liveliness
X	X	X	X	X	X	X
X	X		X	X	X	X
X		X	X	X	X	
X			X	X	X	
	X	X	X	X		X
	X		X	X		X
		X	X	X		
			X	X		
	X					X
						X
X	X	X			X	X
X	X				X	X
X		X			X	X
X					X	X
	X	X			X	X
		X			X	X

Table 1: Implications of combinations of CNF encodings on the theory's properties.

State Coverage The first property refers to the states captured by the boolean theory. That is, when viewed as a policy mapping states to actions, is the policy complete or partial. This property directly follows from the choice of using IMP or not in the encoding. If it is used, then the theory allows for *any* state assignment, and only constrains the space of possible actions when one of the observed states is used.

If IMP is *not* used, then only those states that have been seen in the observation history are possible. Any state configuration that was not seen before would immediately lead to an inconsistency of the theory.

Policy Determinism The next property is *policy determinism*: it holds when at most one action is possible in any assignment that includes a complete state. When this does not hold, the theory represents a non-deterministic policy where a single state may map to multiple actions. This property is guaranteed by AMOA, but can also be achieved through the right combination of the remaining three encoding varieties.

Policy Liveliness The final property is that of *liveliness*: this holds when every complete assignment to the variables in the theory includes at least one true action variable. ALOA by itself can ensure this, as can not using IMP.

Relation of Encodings to Properties The full span of encoding varieties, along with their respective properties, can be found in Table 1. Depending on the application, various properties may be desired. Here, we consider the following:

1. **Not all states covered**: We choose to focus on only capturing a compact policy for those states that were observed in the data $\mathcal{D}$. Relaxing this would realistically require further encoding elements that describe what reasonable

states might look like (perhaps through the use of invariants on the reachable states themselves).
2. **Deterministic policies**: For now we assume that the agent behaviour is deterministic. Given the computation core of model counting, it is natural to consider extending this to *weighted* model counting and allow for a probability distribution over the actions in a particular state, but we leave this extension to future work.
3. **Lively policies**: Again, as a simplifying assumption, we assume that the agent will always execute *some* action in a state. This follows directly from the fact that we consider only the states in the observation data, and every such state has an accompanying action.

These three properties leave us with the final 6 rows in Table 1. Note that the boolean theories that each of these six encodings represent are semantically equivalent: a model of any one theory is a model of all theories. It is these six encodings that we evaluate further in Section 4.

3.3 Inference and Interaction

Unlike the previous approach which compiles a query along with the base theory simultaneously, we appeal to the standard practice in knowledge compilation of amortizing the cost of many queries over a single compilation. The encodings presented in the previous section are used to generate a CNF which is compiled to a d-DNNF representation, which we can then subsequently evaluate and explore interactively. In this work, we focus on two key types of queries:

1. When would you do action `X`?
2. What action would you do when the state looked like `Y`?

Both types of queries can be addressed using the same technique of conditioning on the compiled d-DNNF representation. The assumptions are used to refine the d-DNNF structure, and the query response takes the form of both the resulting d-DNNF after simplification; and the probability distribution of the remaining possibilities for fluents and actions. Both follow standard procedures for d-DNNF processing [5], and are polynomial in the representation size.

Figure 3 shows a portion of the interactive interface we developed. The theory is conditioned by having one fluent assumed to be false, and provides interactive capabilities to explore the remaining models of the theory. Functionality allows us to collapse / expand individual components of the graph, adjust the conditioning via the leaf nodes (both fluent and actions are represented), zoom/pan through the representation, etc. We found this interactive nature to be vital in providing a thorough understanding of the policy.

Figure 4 shows the interface for the aggregate statistics on the likelihood of each fluent and action. Note that the actions are mutually exclusive, and sum to 100%. The percentage represents the proportion of all remaining models where the fluent is true (respectively, the action is executed).

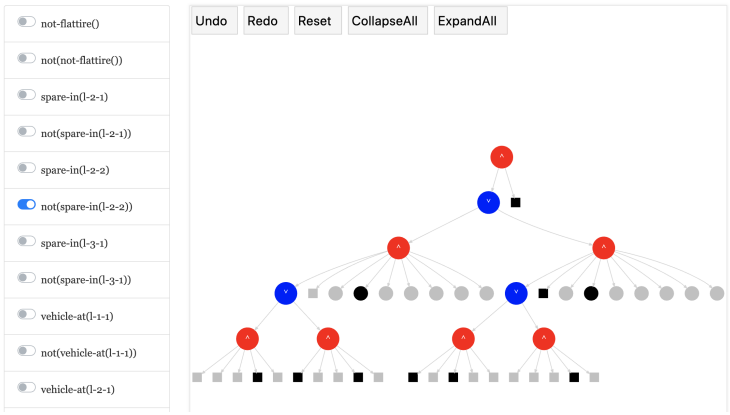
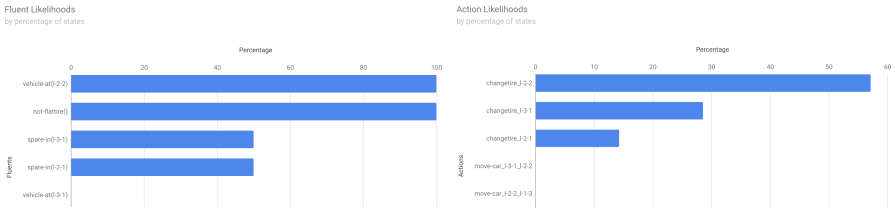


Fig. 3: Conditioned d-DNNF from the interactive interface (one fluent set false).



(a) Assuming `spare-in(l-2-2)` is false. **(b)** Assuming `not-flattire()` is false.

Fig. 4: Screenshots of the (a) fluent and (b) action likelihood interface.

Having both the d-DNNF structure and the aggregate likelihood statistics provides a more holistic view of the policy. The interactive nature is enabled entirely by the amortized view of knowledge compilation – each conditioned operation is computationally efficient given the time invested in computing the d-DNNF in the first place. Next, we explore this computational trade-off in a detailed evaluation.

4 Evaluation

We have two objectives in evaluating the proposed work: (1) to contrast the various encodings that yield the desired properties, and (2) to assess the performance of our approach compared to the previous method for computing a policy and answering queries. We start by describing the domains we consider, and then discuss each evaluation in turn.

4.1 Benchmark Domains

The benchmarks come from two main sources and serve as an illustration that the actual model with which the policy is derived or implemented *does not* impact our ability to capture the behaviour and interact with it. All that is required is the history of state-action pairs that was observed.

domain	problem	$\|\mathcal{F}\|$	$\|\mathcal{D}\|$	TTT	TTF	TFT	TFF	FTT	FFT
blocks	p1	57	8000	201.06	174.44	179.07	**172.42**	178.44	181.60
	p2	162	15462	2118.99	2127.40	**2042.14**	2168.94	2211.26	2203.89
	p3	162	15274	4037.63	3960.20	3981.17	4046.17	**3855.21**	4105.49
elevators	p1	104	17000	175.02	172.38	**151.51**	157.61	155.16	155.76
	p2	295	21268	586.80	**548.14**	638.24	590.61	623.68	559.14
	p3	606	32446	2326.83	2071.53	2390.38	**2037.47**	2198.62	2224.04
tireworld	p1	100	2767	110.90	109.98	111.80	109.22	**105.06**	106.04
	p2	676	5742	1673.18	**1660.00**	1735.03	1735.55	1796.80	1747.09
	p3	2500	8714	24525.47	**21779.88**	24069.43	22773.28	27803.93	27067.64
traffic	p1	8	4321	**626.85**	671.58	725.57	652.81	700.97	700.22
	p2	56	4343	38829.05	35808.71	40226.23	**31715.28**	36813.25	37510.06
	p3	88	4276	126846.23	107324.01	121747.51	110445.90	129135.73	**107180.19**

Table 2: Median compilation times (ms) over 10 trials for the 6 encodings of interest (cf. Section 3.2). The column label for each encoding follows the pattern `(AMOA?)(ALOA?)(OH?)` – e.g., `TFT` indicates that AMOA and OH were used, but not ALOA. We also report the number of fluents ($|\mathcal{F}|$) and size of corpus ($|\mathcal{D}|$). **Bold** indicates best performing encoding.

Planning-Based The first three domains come from the widely available fully observable non-deterministic (FOND) planning benchmark suite. We obtained both the benchmark problems and solutions from the FOND planner PRP [12] using the first few instances in each domain. Note that while the environment itself is non-deterministic, the computed policies are not. That is to say, given a particular state of the world the policy computed by the planner will only ever return a single action (or no action if the state is unreachable).

We generated several traces by repeatedly simulating the found policy from the initial state until the goal condition was reached. Each of the three domains – exploding blocksworld, elevators, and triangle tireworld – had three problems of varying difficulty included.[5]

Deep Reinforcement Learning Based The final benchmark corresponds to a learned policy for controlling the traffic lights at an intersection, as introduced by [18]. The controller chooses between four different actions (see Figure 5) by observing the presence of a vehicle on a cell (lane segment). The three problems correspond to increasing the number of cells per incoming lane (i.e. the higher the problem number, the more segments further from the intersection that are observed). The policy is a neural network learned using standard deep reinforcement learning techniques (which are beyond the scope of this paper).

The states correspond to the traffic conditions on the roads (i.e., the road segments occupancy) while the actions correspond to the various light configurations (advanced left, straight through, etc). The policy was trained to minimize

[5] Despite the naming convention, the problems listed do not correspond to the first three problems of the published benchmark set.

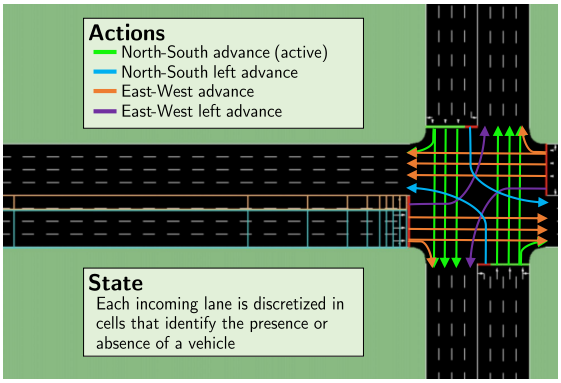


Fig. 5: Example scenario for the traffic-light domain.

the commutative waiting time of vehicles, and an example scenario is shown in Figure 5 with the road segments shown.

For both the method of generating a policy and its representation, there is a large contrast between the planning-based domains and the traffic-light domain. The observed behaviour, however, is the same: discrete state-action pairs.

4.2 Impact of Theory

Given the properties that we desire (cf. Section 3.2), we investigate the efficiency of compiling the 6 theories that capture the properties precisely. We found that the encoding time was negligible for each of the encodings (at most 1-2 seconds for problems that took minutes to compile). Subsequently, we only report on the time to compile the theory.

Table 2 details the median compilation time over 10 runs for each problem and theory combination. We additionally report the number of fluents in the domain ($|\mathcal{F}|$) and the size of the corpus used ($|\mathcal{D}|$). Each of the encodings is described using the pattern `(AMOA?)(ALOA?)(OH?)` – e.g., `TFT` indicates that AMOA and OH were used, but not ALOA.

All of the encodings performed similarly across the domains, which is to be expected given that they are computing the same boolean function and share a common core. However, the configuration `TTF` did stand out as a top performer: either outperforming all of the other encodings or coming in a close second. Note that the ALOA clauses are redundant here, as the encoded theory already indicates which action has occurred for every observed state, and no other state is permitted. This phenomenon of redundant constraints improving the efficiency has long been observed in the constrain programming literature [2].

For the subsequent comparison with a previous approach to computing agent policies, we use this setting, as it was found to generally outperform the others.

4.3 Comparison to Previous Approach

Hayes and Shah [7] similarly demonstrated a system to summarize a robot's behavioral policy as a logical explanation, using a pre-defined vocabulary of boolean predicates. Their system likewise requires the generation of behavior traces detailing each state experienced and action undertaken by the agent: (s_i, a). Given a state s, their system evaluates which boolean predicates are satisfied and which are not: (`True_Predicates`(s), `False_Predicates(`s`)`). From these traces, Hayes and Shah formulate the policy summarization as a set cover problem, which results in the generation of a logical summary for each given behavior. Hayes and Shah's technique uses Quine-McCluskey (QM) to perform the boolean minimization step to solve the set cover problem. In relying on this computationally-intensive boolean minimization process, Hayes and Shah's technique is limited in scale. In comparison, our approach scales by taking advantage of the compact nature of d-DNNF.

We implemented Hayes and Shah's technique for comparison. We apply their QM-based approach to the smallest problems in the four domains presented previously using two queries about when an action would occur and two queries to explain which action will be used given a partial state. These two approaches provide different styles of policy explanations: the QM method listing a series of minterms (i.e., a form of DNF) while our method lists both the compiled d-DNNF structure as well as the normalized action and fluent likelihoods (cf. Figure 4). Here, we assess only the quantitative differences in the compute required.

The results are shown in Table 3. The actual queries used are presented in Appendix A. For both approaches, we use the same preprocessing step to remove static fluents and unused actions from the data. All of the times are listed in milliseconds, and the timeout for QM was set to one hour.

domain	query	QM	compile	condition
blocks	Q1	TO	199.56	41.48
	Q2	TO		37.03
	Q3	TO		34.86
	Q4	TO		43.23
elev	Q5	TO	147.32	37.20
	Q6	TO		29.58
	Q7	TO		44.50
	Q8	TO		32.14
tire	Q9	189.79	133.47	24.44
	Q10	175.38		32.25
	Q11	361.95		25.24
	Q12	191.37		27.13
traffic	Q13	1167.09	499.75	72.85
	Q14	TO		74.69
	Q15	135.05		118.52
	Q16	9.64		75.55

Table 3: Comparison between our approach (**compile** and **condition** columns) with the previous approch (**QM** column). Queries are provided in Appendix A. Times are provided in milliseconds, and a 1-hour time-out (**TO**) was used.

We separate out the compilation and conditioning time for our approach, as the compilation to d-DNNF need only occur once. We found that the conditioning time was indeed very fast, with only one query taking longer than 100ms. In contrast, several of the queries cannot be solved by the QM method within the one hour limit. Ultimately, we found our approach to be a necessary alternative for problems of reasonable size.

As a final analysis, we consider the original trace as DNF directly. We filter the original trace data with a similar preprocessing step to remove static fluents and unused actions, and we treat the remaining state-action pairs as DNF. As this form of DNF is a subset of d-DNNF, the same analysis can be applied. Without the CNF encodings, we found that generally the queries (at least for the modest sized problems and queries considered in Table 3) took roughly twice as long in this brute-force approach. The resulting theories (i.e., after conditioning for each query) ended up being 5-10 times larger. Not only is the representation much larger, but there is no structure contained in the flattened representation. Subsequent conditioning and interaction would be slower given these resultant theories, and we expect further performance degradation for larger problems.

5 Related Work

The AI and planning communities are motivated to design explainable models and techniques to understand existing uninterpretable models. In an effort to replace black box classifiers, Wang et al. infer "OR of ANDs" models for classification [19]. They randomly generate a disjunctive normal form pattern set and then consider any misclassified inputs. If the sample is a false positive, they update their pattern set by decreasing coverage; if the sample is a false negative, they update their pattern set by increasing coverage. Their inference technique is constrained by user-defined priors; for example, such priors may limit the number of clauses and the length of each conjunctive clause. Similar approaches for mining policy explanations from datasets or classifiers have been explored by Rudin et al. [15], McCormick et al. [11], and Cheng et al. [3]. While these approaches enable scalable computation through reliance on strong priors, our approach uses advances in knowledge compilation to facilitate scalability. Further, while these approaches consider individual classification tasks, our approach considers time-series data.

An alternate approach for providing policy explanations is to provide example trajectories to demonstrate how an agent will behave. Amir and Amir create such a policy explanation tool, which extracts 'important' trajectories from simulations of an agent [1]. Huang et al. similarly generate trajectories to communicate to a human how an autonomous system will act in novel situations [8]. While these approaches are better suited to time-series data, it is unclear how trajectories can be effectively used as a communication medium. In contrast, the structure of d-DNNF means that we can readily ask the question, *"When would the agent perform action X?"* and expect a meaningful answer.

Policy summarization is also essential in planning contexts. Humans often want to understand the rationale behind plans, how plans were generated, and how plans compare. Kim et al. provide contrastive explanations for how two plans differ by using a Bayesian probabilistic model to infer linear temporal logic specifications [9]. Seegebarth et al. create a plan explanation prototype which uses first-order logic and requires knowledge-rich plans [16]. Myers developed an approach for communicating differences between plans; their method relies on summarization for Hierarchical Task Networks [14]. However, these approaches are restricted to relatively short plans with specific goals, and build on causal information that we make no presumption of having access to. Additionally, such techniques cannot be readily extended to the scale we are capable of addressing.

6 Conclusion

Facilitating appropriate trust in autonomous systems is a major priority of academia and government. We present one such technique for policy evaluation, which is scalable and agnostic to the autonomous system under consideration. We introduce a method of policy learning from historical agent behaviour through the lens of modern knowledge compilation. We view the historical data as a logical theory, define a family of encodings, and explore a unified subset of these encodings to capture desired properties. In addition to the logic-based framework, we create an interactive experience to explore the compiled policy through iterative refinement and conditioning. The interface additionally provides statistics on the likelihood of each fluent or action given the assumptions.

We empirically evaluated the efficiency of our approach based on modern knowledge compilation, and found it scaled far better than the previous approach dedicated to the same setting. Following the trend in knowledge compilation, a major advantage of our work is the amortized gains: after compiling the data once, we can repeatedly query for various conditions in polynomial time wrt. the size of the compiled representation. This translates to a near-realtime response for each query, and enables the interactive interface. Our work represents a substantial improvement in policy learning and explanation. It additionally serves as a foundation for further exploration of the logical theory.

Future Work The next area to explore is relaxing the assumption of determinism. When the agent behaves probabilistically, the theory changes and so would the compilation. In this case, repeated data cannot be compressed to a single data point. We assert that weighted model counting may enable nondeterminism.

The next property to consider is all states vs. observed states. To generalize beyond the precise states observed, this property must be relaxed. This may have the effect of relegating the statistics with vast amounts of equivalent inconsistent states. Incorporating state invariants from the domain may be helpful.

Finally, we would like to investigate a richer set of queries that include "Why didn't you do X?". We believe that there is a rich space of query types and techniques that can be addressed by operating over the d-DNNF structure directly.

References

1. Amir, D., Amir, O.: Highlights: Summarizing agent behavior to people. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1168–1176. International Foundation for Autonomous Agents and Multiagent Systems (2018)
2. Barták, R.: Theory and practice of constraint propagation. In: Proceedings of the 3rd Workshop on Constraint Programming in Decision and Control. vol. 50 (2001)
3. Cheng, H., Yan, X., Han, J., Hsu, C.W.: Discriminative frequent pattern analysis for effective classification. In: 2007 IEEE 23rd International Conference on Data Engineering. pp. 716–725. IEEE (2007)
4. Darwiche, A.: New advances in compiling CNF to decomposable negation normal form. In: Proceedings of the 16th European Conference on Artificial Intelligence. pp. 318–322 (2004)
5. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002). https://doi.org/10.1613/jair.989
6. Goodman, B., Flaxman, S.: European Union regulations on algorithmic decision-making and a "right to explanation". AI Magazine **38**(3), 50–57 (2017)
7. Hayes, B., Shah, J.A.: 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. pp. 303–312 (2017)
8. Huang, S.H., Held, D., Abbeel, P., Dragan, A.D.: Enabling robots to communicate their objectives. Autonomous Robots **43**(2), 309–326 (2019)
9. Kim, J., Muise, C., Shah, A., Agarwal, S., Shah, J.: Bayesian inference of linear temporal logic specifications for contrastive explanations. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. pp. 5591–5598. AAAI Press (2019)
10. Lagniez, J., Marquis, P.: An Improved Decision-DNNF Compiler. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 667–673 (2017)
11. McCormick, T., Rudin, C., Madigan, D.: A hierarchical model for association rule mining of sequential events: An approach to automated medical symptom prediction. SSRN eLibrary (2011)
12. Muise, C., McIlraith, S.A., Beck, J.C.: Improved Non-deterministic Planning by Exploiting State Relevance. In: The 22nd International Conference on Automated Planning and Scheduling. The 22nd International Conference on Automated Planning and Scheduling (2012)
13. Muise, C., McIlraith, S.A., Beck, J.C., Hsu, E.: DSHARP: Fast d-DNNF Compilation with sharpSAT. In: Canadian Conference on Artificial Intelligence (2012)
14. Myers, K.L.: Metatheoretic plan summarization and comparison. In: ICAPS. pp. 182–192 (2006)
15. Rudin, C., Letham, B., Madigan, D.: Learning theory analysis for association rules and sequential event prediction. The Journal of Machine Learning Research **14**(1), 3441–3492 (2013)
16. Seegebarth, B., Müller, F., Schattenberg, B., Biundo, S.: Making hybrid plans more clear to human users-a formal approach for generating sound explanations. In: Twenty-Second International Conference on Automated Planning and Scheduling (2012)

17. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of reasoning, pp. 466–483. Springer (1983)
18. Vidali, A., Crociani, L., Vizzari, G., Bandini, S.: A deep reinforcement learning approach to adaptive traffic lights management. In: Workshop From Objects to Agents (WOA) (2019)
19. Wang, T., Rudin, C., Doshi-Velez, F., Liu, Y., Klampfl, E., MacNeille, P.: Or's of and's for interpretable classification, with application to context-aware recommender systems. arXiv preprint arXiv:1504.07614 (2015)

A Tested Queries

The following queries correspond to Table 3.

```
Q1: When would you do pick-up_b5_b4?
Q2: When would you do pick-up-from-table_b3?
Q3: What do you do when on-table(b2) & on-table(b3) & detonated(b2) holds?
Q4: What do you do when detonated(b2) & detonated(b3) & on(b3 b2) holds?

Q5: When would you do collect_c2_f2_p2?
Q6: When would you do go-up_e2_f1_f2?
Q7: What do you do when have(c2) holds?
Q8: What do you do when at(f1 p1) holds?

Q9: When would you do move-car_l-2-1_l-3-1?
Q10: When would you do changetire_l-2-1?
Q11: What do you do when vehicle-at(l-2-1) holds?
Q12: What do you do when spare-in(l-2-2) doesn't hold?

Q13: When would you do PHASE_NS_GREEN?
Q14: When would you do PHASE_NSL_GREEN?
Q15: What do you do when car_in_S-G0_0-7 holds?
Q16: What do you do when car_in_S-G0_0-7 & car_in_W-G0_0-7 holds?
```