Explaining Control Policies through Predicate Decision Diagrams

Debraj Chakraborty
chakraborty@fi.muni.cz
Masaryk University
Brno, Czech Republic

Clemens Dubslaff
c.dubslaff@tue.nl
Eindhoven University of Technology
Eindhoven, The Netherlands

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

Jan Kretinsky
jan.kretinsky@tum.de
Masaryk University
Brno, Czech Republic
Technical University of Munich
Munich, Germany

Christoph Weinhuber
christoph.weinhuber@cs.ox.ac.uk
University of Oxford
Oxford, United Kingdom

Abstract

Safety-critical controllers of complex systems are hard to construct manually. Automated approaches such as controller synthesis or learning provide a tempting alternative but usually lack explainability. To this end, learning decision trees (DTs) has been prevalently used towards an interpretable model of the generated controllers. However, DTs do not exploit shared decision making, a key concept exploited in binary decision diagrams (BDDs) to reduce their size and thus improve explainability. In this work, we introduce *predicate decision diagrams* (PDDs) that extend BDDs with predicates and thus unite the advantages of DTs and BDDs for controller representation. We establish a synthesis pipeline for efficient construction of PDDs from DTs representing controllers, exploiting reduction techniques for BDDs also for PDDs.

CCS Concepts

• **Theory of computation → Data structures design and analysis**.

Keywords

Binary decision diagrams, Decision trees, Learning, Explainability, Decision making and control, Strategy synthesis

ACM Reference Format:
Debraj Chakraborty, Clemens Dubslaff, Sudeep Kanav, Jan Kretinsky, and Christoph Weinhuber. 2025. Explaining Control Policies through Predicate Decision Diagrams. In *28th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '25), May 6–9, 2025, Irvine, CA, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3716863.3718049

1 Introduction

Controllers of dynamic systems are hard to construct for a number of reasons: (i) the systems—be it software or cyber-physical systems—are complex due to phenomena such as concurrency, hybrid dynamics, uncertainty, and their steadily increasing sizes; (ii) they are often safety-critical, with a number of complex specifications to adhere to. Consequently, manually crafting the controllers is error-prone. *Automated controller synthesis* or *learning* provide a tempting alternative, where controllers are constructed algorithmically and their correctness is ensured with human participation limited only to the specification process. Numerous approaches have been presented to solve this problem, e.g., exploiting game-theoretic methods to construct a winning strategy (a.k.a. policy) in a game played by the controller and its environment.

Explainability of the automatically constructed controllers (or rather its lack) poses, however, a fundamental obstacle to practical use of automated construction of controllers. Indeed, an unintelligible controller can hardly be accepted by an engineer, who is supposed to maintain the system, adapt its implementation, provide arguments in the certification process, not to speak of legally binding explainability of learnt controllers [1–3]. Unfortunately, most of the synthesis and learning approaches compute controllers in the shape of huge tables of state-action pairs, describing which actions can be played in each state, or of cryptic black boxes. Such representations are typically so huge that they are utterly incomprehensible. An explainable representation should be (i) intuitive with decision entities being small enough to be contained in the human working memory, and (ii) in a simple enough formalism so that the decisions can be read off in a way that relates them to the real states of the system.

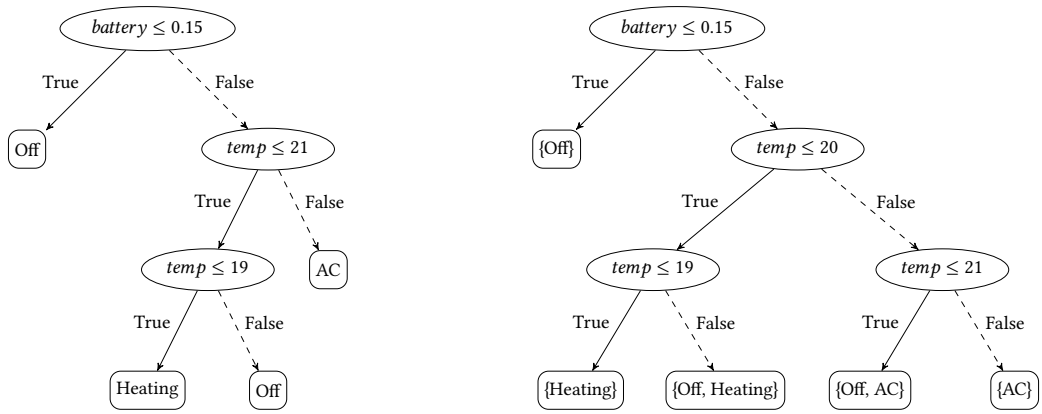
Decision trees (DTs, cf. [50]) have a very specific, even unique position among machine-learning models due to their interpretabilty. As a result, they have become the predominant formalism for explainable representations of controllers [5, 7, 8, 13, 14, 35, 37, 44, 51, 68] over the past decade. Figure 1 shows an example of a policy representation using DTs.

In contrast, reduced ordered *binary decision diagrams (BDDs)* [16] are very popular to represent large objects compactly, exploiting bitwise representation with automatic reduction mechanisms. However, they were barely used as policy representation of controllers due to several advantages of DTs:

1. To represent the controller with non-Boolean state variables using a BDD, the variables are usually *bit-blasted* into a binary encoding. The original controller is then rewritten in terms of these Boolean variables, and a BDD is constructed over them. In contrast, DTs can use more general predicates when arriving

HSCC '25, Irvine, CA, USA

ACM ISBN 979-8-4007-1504-4/2025/05
https://doi.org/10.1145/3716863.3718049



(a) deterministic **(b) permissive**

Figure 1: Examples of DT control policy representations: deterministic control policy (left) and permissive control policy with multiple actions at some states (right).

at a decision. This allows DTs to be smaller, as a single predicate can encompass multiple bits. Additionally, the use of customized predicates makes DTs easier to understand, as they can better incorporate domain knowledge.

❷ The order of predicates along paths in DTs is not fixed as in reduced ordered BDDs. This gives more flexibility and can lead to substantial space savings in DTs.

❸ DT learning allows "don't care" values to be set in any way so that the learnt DT is the smallest. In contrast, BDDs natively represent only fully specified sets, with no heuristic how to resolve the "don't cares" resulting typically in representation larger than necessary.

Still, reduced ordered BDDs exhibit reductions not present in DTs, notably merging isomorphic subdiagrams. This would also be beneficial for explainability of controllers: subdiagram merging (i) reduces the representation size and hence improves explainability following Occam's razor, according to which the best explanation is the smallest among all possible ones [36], and (ii) the psychological complexity towards understanding is also reduced, similar to avoiding code duplication in software [70].

In this paper, we revisit the challenge of marrying concepts from DTs and reduced ordered BDDs towards controller explanations that exhibit advantages of both representations. For this, we build upon the idea of linear decision diagrams [19] and introduce *predicate decision diagrams (PDDs)* as multi-terminal BDDs [34] over bit representations of predicates that inherit all BDD advantages but avoid bit-blasting. PDDs extend DTs and maintain their predicates, profiting from their interpretability, but allow for reduction methods as possible in ordered BDDs.

Specifically, we propose the following approach towards generating reduced ordered PDD from DTs learnt for control policies:

① We collect the predicates mined by the DT representation of a given controller and use them as "variables" in reduced ordered BDDs. This enables use of BDD reductions and mature BDD tooling for PDDs, eliminating issue ❶.

② We apply variable reordering techniques [60] to reduce the size of the PDD, alleviating issue ❷.

③ We use Coudert and Madre's method [23] restricting the BDD support to "care sets", compressing PDDs further, fixing issue ❸.

Notably, due to ① and ②, contradictory branches can arise in PDDs, e.g. by observing contradictory predicates ($temp \leq 19$) and $\neg(temp \leq 20)$ on one decision path. We present a detection and elimination mechanism to remove contradictory branches, leading to *consistent PDDs*, further reducing their size and hence improving explainability of PDDs. We then use our PDD synthesis pipeline, to address the following two research questions:

RQ$_1$ *How do the sizes of PDDs and bit-blasted BDDs relate?*
Following our construction method, PDDs can be represented as (multi-terminal) BDDs, where, however, predicate evaluation does not use bit-blasting as common in classical controller representation via BDDs. The question is whether avoiding bit-blasting can yield more concise representations.

RQ$_2$ *Can controllers be represented more concisely by using ordered PDDs rather than by using DTs?*
Ordered PDDs impose an order on predicates, while DTs can put important decisions earlier in separate branches, leading to smaller tree sizes. However, PDDs allow for merging subdiagrams. The question is which of the complementary reduction mechanisms is more effective.

RQ$_3$ *What is the impact of consistency, reordering, and care-set reduction onto PDD explanations?*
We consider several techniques to reduce the size of PDDs and hence improve their explainability. This research question asks for an ablation study on the impact of these techniques.

Our contributions answer the research questions above and can be summarized as follows:

- We introduce *PDDs*, a BDD data structure allowing for both explainable and small representation of controllers.
- We design an algorithm effectively synthesizing PDD representations, in particular profiting from DT learning and using reasoning modulo DT theory to cater for branch consistency.
- We experimentally show that PDDs and several reduction techniques are beneficial for explaining control policies compared to DT and bit-blasted BDD representations.

We experimentally demonstrate that we almost close the gap between bit-blasted BDD and DT representation sizes. In half of the cases PDDs match the size of DTs or even reduce the size, while retaining the same level of explainability as DTs. Quantitatively, we reduce the gap between BDD and DT sizes on average by 88% on the standard benchmarks. In conclusion, we present PDDs as a viable alternative to DTs, that can offer the best of the both worlds: it is as effective as the DTs as an explainable representation, and can take advantage of the traditional BDD-based tools. Proofs of lemmas and theorems that did not fit into the main paper can be found in the extended version of this paper [20].

2 Related Work

Decision trees (DTs) and similar decision diagram formalisms are prevalently used in explainable AI [4] to create interpretable representations [65–67] of black-box machine learning models. They have been suggested for (approximately) representing controllers and counterexamples in probabilistic systems in [13]. The ideas have been extended to other settings, such as reactive synthesis [14] or hybrid systems [8]. DTs have been used to represent and learn strategies for safety objectives in [54] and to learn program invariants in [35]. Further, DTs were used for representing the strategies during the model checking process, namely in strategy iteration [12] or in simulation-based algorithms [55]. The tool dtControl implements automated synthesis of DTs from controllers [6].

Binary decision diagrams (BDDs) [16] are concisely representing Boolean functions, widely used in symbolic verification [47] and circuit design [49]. In the context of controllers, they have been used to represent planning strategies [22], hybrid system controllers [43, 61], to compress numerical controllers [27], as well as for small representations of controllers through determinization [69]. *Multi-terminal decision diagrams (MTBDDs, [34])*, or more generally referred to as *algebraic DDs (ADDs, [9])*, have been used in reinforcement learning for synthesizing policies [39, 63]. MTBDDs extend BDDs with the possibility to have multiple values in the terminal nodes. *Multi-valued DDs* [48] are extending DDs to allow for multiple decision outcomes in a node over a finite domain rather than only two values. While concise, BDDs are usually not as well-suited for explaining controllers, since states and decisions are usually using standard bit-vector representations that are difficult to map to actual state values and decisions.

Extensions of Decision Diagrams. To avoid bit-blasting and directly enable reasoning on numerical values in decision diagrams, several extensions of BDDs have been proposed. Notably, *edge-valued BDDs* [53] replace bit-level aggregation by summation over integer-labeled edges. Following ideas from satisfiability solving modulo theories (SMT, [10]), the use of BDDs with different theories has been considered in many areas, e.g. to directly speed up predicate abstraction and SMT [18]. Closely related to our work are specific extensions of BDDs where nodes are labeled by predicates from a suitable theory. Here, notable examples are *equational DDs* [33] where nodes might contain equivalences of the form $x = y$, *difference DDs* [52] with predicates of the form $x - y > c$, mainly used in the verification of timed systems, *constrained DDs* [21] extending multi-valued DDs with linear constraints to solve constraint satisfaction problems, and *linear DDs* [19] where nodes are labeled with linear arithmetic predicates. The latter are the closest to our approach towards PDDs, capturing linear predicates and corresponding to *oblique DTs*. The subclass of *axis-aligned DTs* then reflects in our notion of *axis-aligned PDDs*, having the form $x \sim c$ where $\sim$ being a binary comparison relation. Different to our PDDs, linear DDs enforce an implication-consistent ordering on the predicates and use complemented edges to enable constant-time negation operations. We do not opt for complemented edges in PDDs, since they render DDs much more difficult to comprehend. The phenomenon of inconsistent paths in BDDs over predicates from a theory has been already observed in the literature, including sophisticated methods to resolve inconsistencies [19, 32, 33, 52, 64]. Since we aim at explainability of controllers, we focus on simplistic predicates and thus can establish a direct yet effective method of ensuring consistency.

Dubslaff et al. [29] present *template DDs* (TDDs) to model and improve explainability of self-adaptive systems and DTs learnt from controllers. Following the concept of procedures, templates in TDDs can be instantiated with decision-making patterns, leading to hierarchical (unordered) multi-valued DDs. Similar to our approach, DT predicates are interpreted as Boolean variables and sharing of decision making yields size and cognitive load reductions, both improving explainability.

Recently, there has been research on learning DDs for classification of training data. Cabodi et al. [17] propose a SAT-based approach towards a BDD representation of a DT. Hu et al. [40] use SAT- and a lifted MaxSAT-based model to learn optimal BDDs with bounded depth. Florio et al. [31] propose a mixed-integer linear programming method to learn optimal (not necessarily binary) DDs.

3 Preliminaries

For a set X, we denote its power set by 2^X. A *total order* over a finite set X is a bijection $\tau\colon X \to \{1, \dots, |X|\}$ where $|X|$ is the number of elements in X. Let us fix a set of *variables* Var and a *variable domain* $\mathbb{D}$. Canonical domain candidates are the boolean domain $\mathbb{B} = \{0, 1\}$, natural numbers $\mathbb{N}$, rational numbers $\mathbb{Q}$, and real numbers $\mathbb{R}$. An *evaluation* is a function $\epsilon\colon Var \to \mathbb{D}$; the set of evaluations is denoted by $Eval(Var, \mathbb{D})$. An (axis-aligned) *predicate* is of the form $x \sim c$ where $x \in Var$ is a variable, $c \in \mathbb{D}$, and $\sim \in \{=, \geqslant, >\}$ is a comparison relation over $\mathbb{D}$. The set of predicates is denoted by $\mathbb{P}(Var, \mathbb{D})$. A *simple* predicate is of the form $x > 0$ with $x \in Var$, denoted by x for brevity[1]. A predicate $x \sim c \in \mathbb{P}(Var, \mathbb{D})$ is *satisfied* in an evaluation $\epsilon\colon Var \to \mathbb{D}$, denoted $\epsilon \vDash (x \sim c)$, iff $\epsilon(x) \sim c$. We introduce propositional formulas over $\mathbb{P}(Var, \mathbb{D})$ by the grammar $\phi ::= \texttt{true} \mid p \mid \neg\phi \mid \phi \wedge \phi$ where p ranges over $\mathbb{P}(Var, \mathbb{D})$. The satisfaction relation $\vDash$ extends towards formulas by $\epsilon \vDash \neg\phi$ iff $\epsilon \nvDash \phi$, and $\epsilon \vDash \phi_1 \wedge \phi_2$ iff $\epsilon \vDash \phi_1$ and $\epsilon \vDash \phi_2$.

Bit-Blasting. Let S be a finite set of states with $|S|$ elements and $n = \lceil \log_2 |S| \rceil$. We define bit-blasting as an injective mapping $\beta\colon S \to \mathbb{B}^n$, which is constructed in two steps: First, we enumerate states by a total order $\tau\colon S \to \{1, \dots, |S|\}$. Second, for each $s \in S$, we assign the binary representation $\beta(s) = (b_{i,1}, b_{i,2}, \dots, b_{i,n})$ where $(b_{i,1}, \dots, b_{i,n})$ is the n-bit encoding of the integer $i = \tau(s) - 1$. Bit-blasting converts each high-level state into a unique bit vector.

[1] Therefore, we consider variables to be contained in the set of predicates, i.e., $Var \subseteq \mathbb{P}(Var, \mathbb{D})$, interpreted as simple predicates.

3.1 Policies

Definition 3.1 (Policy). For an operational model with a set of states S and actions Act, a *policy* $\sigma\colon S \to 2^{Act}$ selects for every state $s \in S$ a set of actions $\sigma(s) \subseteq Act$.

Note that this definition allows *permissive* (or *nondeterministic*) policies that can provide multiple possible actions for a state. A policy σ is *deterministic* if $|\sigma(s)| = 1$ for all $s \in S$.

We assume that the set of states S is structured, i.e., every state is a tuple of values of state variables. In other words, states are not monolithic (e.g., a simple numbering) but there are multiple factors defining it. Each of these factors is defined by a variable from the set of variables Var. Therefore, we can alternatively view a policy as a (partial) function $\sigma\colon Eval(Var, \mathbb{D}) \to 2^{Act}$ that maps an evaluation $\epsilon\colon Var \to \mathbb{D}$ to a set of actions $\sigma(\epsilon) \subseteq Act$.

Example 3.2. Consider the policy in Figure 2a. There are two state variables $Var = \{x, y\}$. The set of actions is $Act = \{a, b\}$.

Note that a policy is usually partial, i.e., not all evaluations are in S, e.g., standing for states not reachable in the system model.

3.2 Decision Trees

Definition 3.3 (DT). A *decision tree (DT)* T is defined as follows:

- T is a rooted full binary tree, meaning every node either is an inner node and has exactly two children, or is a leaf node and has no children.
- Every inner node v is associated with a predicate in $\mathbb{P}(Var, \mathbb{D})$.
- Every leaf ℓ is associated with an output label a_ℓ.

Numerous methods exist for learning DTs, e.g., CART [15], ID3 [56], and C4.5 [57]. These algorithms all evaluate different predicates by calculating some *impurity measure* and then greedily pick the most promising to split the dataset on that predicate into two halves. This is recursively repeated until a single action in the dataset is reached, constituting a leaf of the resulting binary tree.

A DT represents a function as follows: an evaluation ϵ is evaluated by starting at the root of T and traversing the tree until we reach a leaf node ℓ. Then, the label of the leaf a_ℓ is our prediction for the input ϵ. When traversing the tree, at each inner node v we decide at which child to continue by evaluating the predicate α_v with ϵ. If the predicate evaluates to true, we pick the left child, otherwise, we pick the right one.

When we represent a (permissive) policy $\sigma\colon S \to 2^{Act}$ with a DT, our input data is the set of states S and an output labels describe a subset of actions Act. Unlike algorithms in machine learning, our objective is to learn a DT that represents the policy on all data points available from the controller. Therefore, we do not stop the learning based on a stopping criterion but overfit to completely capture the training data. *Safe early stopping* [7] can be used to produce smaller DTs that do not represent the full permissive policy σ but a deterministic σ' such that $\sigma'(s) \subseteq \sigma(s)$ for all $s \in S$. Figure 2b depicts a DT representing the policy of Example 3.2.

3.3 Binary Decision Diagrams

Definition 3.4 (BDD). A (reduced ordered multi-terminal) *binary decision diagram (BDD)* is a tuple $\mathscr{B} = (N, Var, Act, \lambda, \pi, \mathsf{succ}, r)$ where N is a finite set of nodes, $\lambda\colon N \to Var \cup 2^{Act}$ is a function labeling decision nodes $D \subseteq N$ by Boolean variables Var and action nodes $N \setminus D$ by sets of actions Act, π a total order over Var, $\mathsf{succ}\colon D \times \mathbb{B} \to N$ is a decision function, and $r \in N$ is a root node such that $\mathscr{B}$ is

- *ordered*, i.e., for all $d \in D$ and $b \in \mathbb{B}$ with $\mathsf{succ}(d, b) \in D$ we have $\pi\big(\lambda(d)\big) < \pi\big(\lambda(\mathsf{succ}(d, b))\big)$, and
- *reduced*, i.e., all decisions $d, d' \in D$ are *essential* ($\mathsf{succ}(d, 0) \neq \mathsf{succ}(d, 1)$) and $\mathscr{B}$ does not contain isomorphic subdiagrams.

A π-BDD is a BDD that is ordered w.r.t. the variable order π. A *path* in $\mathscr{B}$ is an alternating sequence $d_0, b_0, d_1, \ldots, d_k \in (D \times \mathbb{B})^* \times N$ where $d_{i+1} = \mathsf{succ}(d_i, b_i)$ for all $i < k$. Given a node $n \in N$, we denote by $\mathsf{Reach}(n) \subseteq N$ the nodes reachable by a path from n. In the following, we assume all nodes N in $\mathscr{B}$ to be reachable from root r. By $\mathscr{B}_n$ we denote the sub-BDD of $\mathscr{B}$ that comprises nodes $\mathsf{Reach}(n)$ and root n. Note that reducedness ensures that for each subset of actions $A \subseteq Act$ there is at most one node $a \in N \setminus D$ that is labeled by A, i.e., $\lambda(a) = A$.

Semantics of BDDs. A BDD over Var and Act represents a function $f\colon Eval(Var, \mathbb{B}) \to 2^{Act}$ as follows: every evaluation ϵ is evaluated by starting at the root r and traversing the graph until we reach a terminal action node v_t. Then $f(\epsilon) = \lambda(v_t)$. When traversing, at each decision node $d \in V$ we decide at which child to continue by checking the value of $\epsilon(\lambda(d))$: if $\epsilon(\lambda(d)) = 1$, we pick $\mathsf{succ}(d, 1)$ as the next node. Otherwise, we pick $\mathsf{succ}(d, 0)$ as the next node. Formally, given a node n in $\mathscr{B}$, we define the function $f = [\![\mathscr{B}_n]\!]$ recursively through evaluations $\epsilon\colon Var \to \mathbb{B}$:

$$[\![\mathscr{B}_n]\!](\epsilon) = \begin{cases} \lambda(n) & \text{if } n \in N \setminus D \\ [\![\mathscr{B}_{\mathsf{succ}(n, \epsilon(\lambda(n)))}]\!](\epsilon) & \text{if } n \in D \end{cases}$$

To represent a policy $\sigma\colon S \to 2^{Act}$ using a BDD, we treat it as a function f_σ over evaluations as above: we represent each state $s \in S$ as a bitvector $\bar{s}$ and define f_σ by $f_\sigma(\bar{s}) = \sigma(s)$.

While BDDs are a useful data structure to represent policies, they have a few disadvantages compared to the DTs. Firstly, nodes in BDDs correspond to simple predicates in DTs, making them less human-interpretable compared to DTs. In contrast, DTs supports a richer variety of predicates and variable domains (even exceeding the class of axis-aligned predicates we focus on in this paper). Secondly, BDDs have a canonical form that ensures a unique representation as they are ordered and reduced. This facilitates efficient comparison and manipulation of BDDs. In contrast, DTs allow flexible ordering, which might allow for smaller lengths of decision sequences, possibly easier to understand. Thirdly, a DT can be generated from a partial policy, ignoring some inputs towards smaller representations. In contrast, existing BDD constructions are not directly optimized for small representations of partial functions.

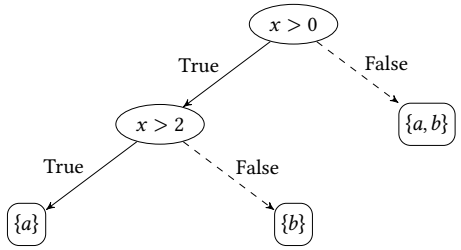
4 Predicate Decision Diagrams

In this section, we propose *predicate decision diagrams (PDDs)* as an instance of linear DDs [19] that are extended to represent control policies. PDDs combine concepts from DTs and BDDs to benefit from both formalism's advantages.

Definition 4.1. A *predicate decision diagram (PDD)* over Var and $\mathbb{D}$ is a tuple $\mathscr{P} = (N, Var, Act, \lambda, \mathsf{succ}, r)$ where N is a finite set of nodes, $\lambda\colon N \to \mathbb{P}(Var, \mathbb{D}) \cup 2^{Act}$ is a function that labels decision

x	y	actions
0	0	$\{a, b\}$
1	0	$\{b\}$
2	0	$\{b\}$
3	1	$\{a\}$

(a)



(b)

Figure 2: An example of a policy in the form of a lookup table (left), and the corresponding decision tree (right).

nodes $D \subseteq N$ by predicates over Var and $\mathbb{D}$ and action nodes $N \setminus D$ by sets of actions Act, $\mathsf{succ} : D \times \mathbb{B} \to N$ is a decision function, and $r \in N$ is a root node.

Notice the similarities with the BDD definition (see Definition 3.4). While in BDDs decision nodes are labeled by variables, they are labeled by predicates in PDDs. Throughout this section, let us fix a PDD $\mathscr{P}$ as above. We inherit BDD notations as expected and call $\mathscr{P}$

- *deterministic* if all reachable action nodes are labeled by singletons, i.e., $a \in N \setminus D$ implies $|\lambda(a)| = 1$.
- *ordered* for a total order π over $\lambda(D)$ if for all $d \in D$ and $b \in \mathbb{B}$ with $\mathsf{succ}(d, b) \in D$ we have $\pi\big(\lambda(d)\big) < \pi\big(\lambda(\mathsf{succ}(d, b))\big)$.
- *reduced* if all decisions $d, d' \in D$ are *essential* (i.e. $\mathsf{succ}(d, 0) \neq \mathsf{succ}(d, 1)$) and $\mathscr{P}$ does not contain isomorphic subdiagrams (i.e., $\mathscr{P}_d \cong \mathscr{P}_{d'}$ implies $d = d'$).

To distinguish between components of a PDD $\mathscr{P}$ or BDD $\mathscr{B}$, we occasionally add suffixes, e.g., $\lambda_{\mathscr{B}}$ to indicate the labeling function of $\mathscr{B}$. Note that our definition of PDDs covers the standard notions of DTs and BDDs: A DT is a PDD where the underlying graph has a tree structure: for all $n, n' \in N$ with $n' \notin \mathsf{Reach}(n)$ and $n \notin \mathsf{Reach}(n')$ we have $\mathsf{Reach}(n) \cap \mathsf{Reach}(n') = \emptyset$. A BDD is a reduced ordered PDD where all predicates are simple.

Semantics of PDDs. Given a node n in $\mathscr{P}$, we define a policy $[\![\mathscr{P}_n]\!]$ recursively through evaluations $\epsilon : Var \to \mathbb{D}$:

$$[\![\mathscr{P}_n]\!](\epsilon) = \begin{cases} \lambda(n) & \text{if } n \in N \setminus D \\ [\![\mathscr{P}_{\mathsf{succ}(n,0)}]\!](\epsilon) & \text{if } n \in D \text{ and } \epsilon \not\models \lambda(n) \\ [\![\mathscr{P}_{\mathsf{succ}(n,1)}]\!](\epsilon) & \text{if } n \in D \text{ and } \epsilon \models \lambda(n) \end{cases}$$

Two PDDs $\mathscr{P}$ and $\mathscr{P}'$ are *semantically equivalent* iff $[\![\mathscr{P}]\!](\epsilon) = [\![\mathscr{P}']\!](\epsilon)$ for all evaluations $\epsilon : Var \to \mathbb{D}$. A path $d_0, b_0, d_1, \ldots, d_k \in (D \times \mathbb{B})^* \times N$ is *consistent* with an evaluation $\epsilon \in Eval(Var, \mathbb{D})$ if for all $i \leqslant k$ we have $\epsilon \models \lambda(d_i)$ iff $b_i = 1$. We call a PDD *consistent* if for all paths from the root to an action node there is a consistent evaluation. Note that different to BDDs, the essentiality of reduced PDDs does not propagate on a semantic level, i.e., a decision node can have different successors but still may be irrelevant for the overall represented policy. For instance, if we would replace $\{a, b\}$ in Figure 2 by $\{b\}$, the decision $x > 0$ becomes irrelevant for the policy even though it has two different successors ($x > 2$ and $\{b\}$). The reason is in the possible semantic dependence of predicates, leading to the policy yielding $\{a\}$ if $x > 2$ and $\{b\}$ otherwise, independent of the decision $x > 0$. Consistency however ensures that after reaching a predicate node there are always evaluations for both cases, satisfaction and unsatisfaction of the predicate.

Algorithm 1 Pdd2Bdd($\mathscr{P}$,γ,π): compile PDD $\mathscr{P}$ to the π-BDD $\mathscr{B}$ w.r.t. γ

Input: PDD $\mathscr{P} = (N, Var, Act, \lambda, \mathsf{succ}, r)$, bijection $\gamma : PVar \to \lambda(D)$, $PVar$ order π
Output: A π-BDD $\mathscr{B}$

```
if λ(r) ⊆ Act then                                   ▷ action set labeled root
    return B = ({r}, PVar, Act, {(r, λ(r))}, π, ∅, r)
else                                                 ▷ predicate labeled root
    B_0 ← Pdd2Bdd(P_succ(r,0), γ, π)                 ▷ compile 0-successor PDD
    B_1 ← Pdd2Bdd(P_succ(r,1), γ, π)                 ▷ compile 1-successor PDD
    return B = ITE(γ^-1(λ(r)), B_1, B_0)
```

4.1 From PDDs to BDDs and Back

Taking on a Boolean perspective on predicates yields equivalence classes for evaluations that satisfy the same predicates. Then, predicates can be seen as Boolean variables evaluated over equivalence class containment. Therefore, we now investigate connections between PDDs and BDDs [9, 34] and that enable mature theory and tool support of BDDs also for PDDs.

4.1.1 BDD Encoding. Consider a PDD $\mathscr{P} = (N, Var, Act, \lambda, \mathsf{succ}, r)$ over Var and $\mathbb{D}$. Let $PVar$ denote a set of Boolean *predicate variables* for which there is a bijection $\gamma : PVar \to \lambda(D)$ we call *predicate bijection*. Further, define the *γ-lifting* of an evaluation $\epsilon : Var \to \mathbb{D}$ over Var as the Boolean evaluation $\gamma^\epsilon : PVar \to \mathbb{B}$ where for all $x \in PVar$:

$$\gamma^\epsilon(x) \;=\; \begin{cases} 1 & \text{if } \epsilon \models \gamma(x) \\ 0 & \text{otherwise} \end{cases}$$

We then say that a BDD $\mathscr{B}$ over $PVar$ is *equivalent modulo γ* to a PDD $\mathscr{P}$ iff $[\![\mathscr{B}]\!](\gamma^\epsilon) = [\![\mathscr{P}]\!](\epsilon)$ for all $\epsilon : Var \to \mathbb{D}$. Any BDD $\mathscr{B}$ over $PVar$ and γ trivially exhibits an equivalent PDD $\gamma(\mathscr{B})$ that arises from $\mathscr{B}$ by changing the labeling of each decision node $d \in D_{\mathscr{B}}$ to $\gamma(\lambda_{\mathscr{B}}(d))$. A path in $\mathscr{B}$ is *predicate consistent* w.r.t. γ if its corresponding path in the PDD $\gamma(\mathscr{B})$ is consistent. Likewise, $\mathscr{B}$ is *predicate consistent* w.r.t. γ if $\gamma(\mathscr{B})$ is consistent. Note that we consider BDDs to be reduced and ordered, hence $\gamma(\mathscr{B})$ is a reduced and ordered PDD. The other way around, any reduced ordered PDD $\mathscr{P}$ directly provides a BDD by defining Boolean variables $PVar$ for each predicate in $\mathscr{P}$ towards a predicate bijection γ where each labeling of decision nodes $d \in D$ are replaced by a predicate variable $\gamma^{-1}(\lambda(d))$.

The main advantages of BDDs, namely canonicity and concise representation, cannot directly be expected for PDDs. Our goal is

Algorithm 2 PConsistency($\mathscr{B}, \gamma, \phi$): Turn a π-BDD $\mathscr{B}$ into a π-BDD $\mathscr{B}'$ predicate consistent w.r.t. predicate bijection γ

Input: π-BDD $\mathscr{B} = (N, PVar, Act, \lambda, \pi, \mathsf{succ}, r)$, $\gamma\colon PVar \to \mathbb{P}(Var, \mathbb{D})$, context predicate formula ϕ over $\mathbb{P}(Var, \mathbb{D})$
Output: A π-BDD $\mathscr{B}'$

if $\lambda(r) \subseteq Act$ **then** ▷ actions (base)
  return $\mathscr{B}' = (\{r\}, PVar, Act, \{(r, \lambda(r))\}, \pi, \emptyset, r)$
else ▷ predicates (recursion step)
  if $\phi \wedge \gamma(\lambda(r))$ is satisfiable **then**
    $\mathscr{B}'_1 \leftarrow$ PConsistency$(\mathscr{B}_{\mathsf{succ}(r,1)}, \gamma, \phi \wedge \gamma(\lambda(r)))$
    if $\phi \wedge \neg\gamma(\lambda(r))$ is satisfiable **then**
      $\mathscr{B}'_0 \leftarrow$ PConsistency$(\mathscr{B}_{\mathsf{succ}(r,0)}, \gamma, \phi \wedge \neg\gamma(\lambda(r)))$
      return $\mathscr{B}' = \mathrm{ITE}(\lambda(r), \mathscr{B}'_1, \mathscr{B}'_0)$
    else
      return $\mathscr{B}' = \mathscr{B}'_1$
  else
    $\mathscr{B}'_0 \leftarrow$ PConsistency$(\mathscr{B}_{\mathsf{succ}(r,0)}, \gamma, \phi \wedge \neg\gamma(\lambda(r)))$
    return $\mathscr{B}' = \mathscr{B}'_0$

hence to enable these desirable properties also for PDDs, including possibilities to exploit mature theory and broad tool support available for BDDs. While ordering, reducedness, and (predicate) consistency are obviously maintained by the transformations above, our main application concerns PDDs in form of DTs, which are a priori neither ordered nor reduced.

Algorithm 1 compiles a PDD $\mathscr{P}$ into a BDD $\mathscr{B}$ that is equivalent to $\mathscr{P}$modulo γ. The algorithm recursively traverses the PDD until reaching an action node (see Section 4.1) which is directly returned as BDD with an action node as root. Here, we rely on the ITE operator (if-then-else) to implement the interpretation of the root predicate $\lambda(r)$ by the BDD predicate variable $x = \gamma^{-1}(\lambda(r))$. The ITE operator is standard for BDDs and implements the Shannon expansion of $[\![\mathscr{B}]\!]$, i.e., return $[\![\mathscr{B}_1]\!]$ if x holds and $[\![\mathscr{B}_0]\!]$ otherwise.

Example 4.2. Figure 3a shows an application of Algorithm 1 on the PDD in Figure 2b according to the variable order $\pi = (p_1, p_0)$ over predicate variables $X = \{p_0, p_1\}$ with $\gamma(p_0) = (x > 0)$ and $\gamma(p_1) = (x > 2)$. Note that the resulting π-BDD is not predicate consistent, since $x > 2$ implies also $x > 0$ and hence, $(x > 2) \wedge \neg(x > 0)$ is unsatisfiable, leading the 0-successor of $x > 0$ to induce inconsistent paths (in red).

Lemma 4.3. *Given a PDD $\mathscr{P}$, a predicate bijection γ for $\mathscr{P}$ over variables PVar, and a total order π over PVar, Pdd2Bdd$(\mathscr{P}, \gamma, \pi)$ (see Algorithm 1) returns a π-BDD that is equivalent modulo γ to $\mathscr{P}$.*

4.1.2 Predicate Consistent BDDs. The BDD returned by Algorithm 1 does not have to be predicate consistent. Since PDDs are a priori not ordered, inconsistent orders of decisions might be enforced by the order π along Pdd2Bdd performs its compilation (see Figure 3a).

We now present a simple yet effective method to prune inconsistent branches in BDDs while maintaining their semantics in Algorithm 2. Here, we rely on satisfiability solving over predicate formulas, an instance of a classical problem for state-of-the-art SMT solvers. The algorithm traverses the BDD and keeps track of the visited predicates that were considered to be satisfied or not. The latter is achieved by a *context predicate formula* ϕ that is a conjunction over all past satisfaction decisions on predicates. If one of the predicates assigned to the current decision node or its negation is unsatisfiable within the context, then the opposite decision is taken and the corresponding decision node is returned. Note that it cannot be that a predicate and its negation are both unsatisfiable.

Lemma 4.4. *For a given π-BDD $\mathscr{B}$ over PVar and a predicate bijection $\gamma\colon PVar \to \mathbb{P}(Var, \mathbb{D})$, PConsistency$(\mathscr{B}, \gamma, \mathtt{true})$ (see Algorithm 2) returns a predicate consistent π-BDD $\mathscr{B}'$ that has the same semantics modulo γ as $\mathscr{B}$, i.e., $[\![\mathscr{B}]\!](\gamma^\epsilon) = [\![\mathscr{B}']\!](\gamma^\epsilon)$ for all evaluations $\epsilon\colon Var \to \mathbb{D}$.*

Example 4.5. Figure 3b shows how to create a predicate-consistent π-BDD from the π-BDD in Figure 3b. The left p_0-node (labeled by the predicate "$x > 0$") can be safely removed and the 1-successor of the root is redirected to the leftmost a-node (i.e., by removing the greyed part and introducing the blue branch).

Combining Lemma 4.3 and Lemma 4.4, we obtain a method for predicate consistent BDD representations of a given PDD and can benefit from standard BDD techniques to represent control policies:

Theorem 4.6. *Given a PDD $\mathscr{P}$ and a total order π on its predicates there is a consistent reduced π-PDD represented by a predicate consistent BDD semantically equivalent to $\mathscr{P}$.*

4.2 Towards PDD Explanations

Due to BDDs admitting a canonical representation w.r.t. to a given variable order and being reduced, BDDs admit minimal functional representations. This renders BDDs also suitable for explanations: According to Occam's razor, the best explanation for a phenomenon is the most simple one amongst all possible explanations (cf. [36]). Still, minimality heavily relies on fixing a variable order and a better explanation is possible (i.e., a smaller BDD representation) for a different variable order.

4.2.1 Variable-order Optimization. Compared to DTs and hence general PDDs, enforcing a variable order comes at its price, possibly countering concise representation. The latter phenomenon is well-known in the field of BDDs where switching to variable trees instead of orders can provide exponentially more compact representations [24]. Also different variable orders already might yield even more concise representations.

Thanks to our BDD representation of consistent reduced ordered PDDs (see Theorem 4.6), the advantages of BDDs carry over to PDDs, ensuring concise representations for a given variable order. Fortunately, this holds also for well-known techniques applicable on BDDs to mitigate variable order restrictions. Deciding whether a given variable order is suboptimal is an NP-complete problem [11]. To this end, heuristics have been developed to find good variable orders (see, e.g., [28, 59]). One prominent method that is applicable on already constructed BDDs is provided by Rudell's sifting method [60]. The idea is to permute adjacent variables in the variable order by swapping levels of decision nodes in-place, leading to a permutation that provides smaller diagram sizes. The result of one swap operation applied on the (consistent reduced ordered) PDD of Figure 2 on variables for predicates $x > 0$ and $x > 2$ can be seen in Figure 3. Here, the resulting BDD increases in size and thus, such a swap might be not considered as favorable towards a better variable ordering. However, sifting minimizes the BDDs taking on

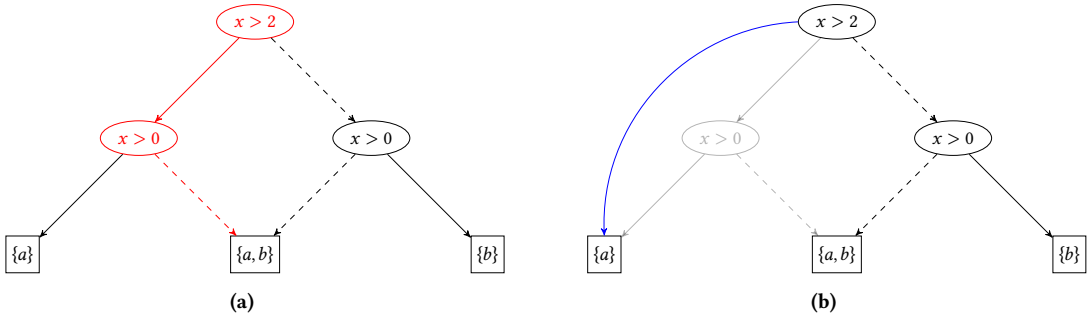


Figure 3: Compilation of the DT in Figure 2b to a (predicate inconsistent) BDD (left) using Algorithm 1. A predicate consistent BDD (right) can be created by replacing the inconsistent part by the blue edge using Algorithm 2.

a Boolean interpretation of the variables. If the BDD represents a PDD (see Lemma 4.3) the semantic interpretation of predicates is not taken into account. To this end, sifting might introduce inconsistent branches, i.e., turn even a predicate consistent BDD into an inconsistent one. A similar observation has been already drawn when reordering linear DDs [19]. In our previous example of the swap operation onto Figure 2, this phenomenon can be seen in Figure 3, where $\neg(x > 0)$ cannot be satisfied after deciding for $x > 2$. Hence, BDD representations for PDDs can be further reduced after sifting using our consistency transformation (Algorithm 2).

4.2.2 Care-set Reduction. In practice, controller policies are usually provided by partial functions, where decisions in certain states are not relevant and can be chosen arbitrarily. DT learning algorithms for their explanation exploit these to reduce their size. Differently, PDDs and BDDs represent total functions, not exploiting the full potential for even more concise representations. Fortunately, this application is well-understood in the field of BDD-based symbolic verification, where fixed points over partial state domains are crucial towards performant verification algorithms. For this, Coudert and Madre introduced the *restrict operator* [23] that minimizes a given BDD according to a *care set* of variable evaluations. Technically, nodes whose semantics agree in all evaluations in the care set are merged into existing nodes, exploiting the sharing in BDDs. In our setting, we apply the restrict operator on the domain of the controller function as care set. We then obtain a reduced ordered consistent PDD that has a smaller size and yields the same outcomes for all original state-action pairs used to learn the (total control policy representing) DT. The outcomes for other state evaluations than in the domain of the state-action pairs are used to reduce the PDD, leading to a total policy representation that is likely to differ from the one of the original DT. Note that the restrict operation does not introduce new inconsistencies as it does not change the predicate order in PDDs and only removes nodes on decision paths.

5 Experimental Evaluation

In this section, we describe our experimental setup and compare several kinds of controller representations: BDDs with bit-blasting (bbBDDs), DTs, and reduced ordered PDDs (including reordered and consistent variants). Our experimental results show that, i) PDDs are almost as effective as a controller representation as DTs (answering $\mathbf{RQ_2}$), ii) PDDs are more compact and explainable than bbBDDs (answering $\mathbf{RQ_1}$), and iii) our reduction methods for PDDs are effective for controller representation (answering $\mathbf{RQ_3}$).

Synthesis Pipeline. We implemented our approach in the tool dtControl [7] using a python wrapper to the BDD library BuDDy [46]. Figure 4 shows the pipeline we used to construct consistent reduced ordered PDDs from tabular policies. DTs are learnt from given controller policies (with axis-aligned predicates using entropy as the impurity measure), compiling them into BDDs by Algorithm 1, optimizing their variable ordering through reordering, and ensuring their predicate consistency before applying care-set reduction.

BDD Implementations. We report on two different kinds of BDD representations for policies: bit-blasted BDDs (bbBDDs) and BDD representations for (reduced ordered) PDDs. The former follow the classical approach of a binary encoding of state variables' domains and constructing a BDD representation for the (partial) policy $\sigma : \mathit{Eval}(\mathit{Var}, \mathbb{D}) \to 2^{\mathit{Act}}$ from the data set. The variable order for bbBDDs is initialized by replacing the order of state variables by the encoded bit vector blocks, followed by variable reordering through sifting until convergence.

For the BDDs constructed via our PDD pipeline, we initially chose a predicate variable order arising from a breadth-first traversal of the input DT. This ensured that the variable ordering used for BDD construction closely mirrors the structure of the learned DT.

Note that towards a more fair comparison we do not include the multiplicities of action nodes in DTs and only count decision nodes. Further, to ensure a more fair comparison towards explainability, bbBDDs are not encoding actions binary and not choosing a complemented edge representation as done, e.g., in [7].

Benchmarks. For evaluation, and to align with the existing comparison of DTs and bbBDDs, we use the standard benchmarks of the tool dtControl [7]. The benchmarks contain permissive policies for cyber-physical systems exported from Scots [61] and UppAal [25] and from the Quantitative Verification Benchmark Set [38], also including models from the Prism Benchmark Suite [45]. These case studies were solved using Storm [26] and exported as JSON files. The policies given by Scots or UppAal are permissive. Storm only exports deterministic policies.

Experiment Setup. Our experiments were executed on a desktop machine with the following configuration: one 4.70 GHz CPU (AMD Ryzen™ 7 PRO 5750G) with 16 processing units (virtual cores), 66 GB RAM, and 22.04.1-Ubuntu as operating system.

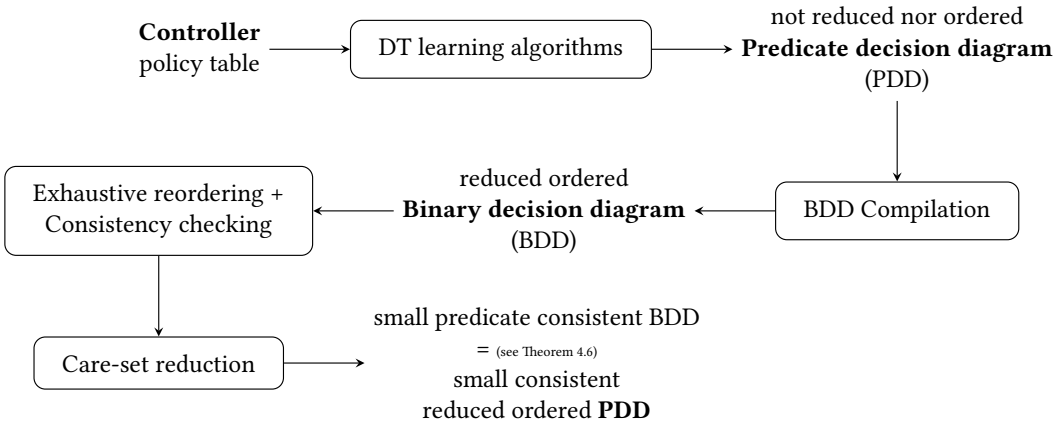


Figure 4: PDD synthesis pipeline used in the experiments

Extensions. Our implementation also supports PDDs with linear predicates, i.e., where the decision node could contain an inequality of a linear combination of variables. We restricted ourselves to axis-aligned predicates (that use only one variable in the decision nodes) as they are easier to interpret. Further, in our initial experiments, we found that using linear predicates (as in the case of linear DDs [19]) also resulted in less sharing and thus larger PDDs. For our pipeline, the reductions can be also put into different order. However, we here chose the order that led to the best results on average.

5.1 Concise PDD Representation

Towards answering $\mathbf{RQ_2}$ and $\mathbf{RQ_1}$, we conducted several experiments those results are shown in Table 1. Here, we investigated the representation gaps between bbBDDs, PDDs, and DTs.

5.1.1 Gap Between Decision Diagrams and DTs. Using the classical BDD-based representation of control policies through bit-blasted BDDs (bbBDDs), there is a well-known gap between bbBDDs and DT sizes [7]. The key question is whether PDDs can be used instead of DTs without efficiency drop (increase in size). For this, we first report on the degree of closing the gap defined as

$$R_{gap} = (bbBDD - PDD) \,/\, (bbBDD - DT)$$

where *bbBDD*, *DT*, and *PDD* denote the number of decision nodes in the smallest bbBDD that we found, the DT generated using dtControl, and the reduced ordered concise PDDs generated using our synthesis pipeline, respectively. An R_{gap} value closer to 1 means a PDD improves almost to the level of DTs in term of size, while a value closer to 0 means a PDD remains as bad as bbBDDs. The computation excludes the cases (three in total) where the DTs are larger than bbBDDs and the PDDs are smaller (*10rooms*, *traffic_30m*, and *elevators.a-11-9*), i.e., where PDDs are even smaller than the DTs. We bridge the gap between BDD and DT sizes on average by 88%. Consequently, we claim that PDD can safely replace DT without serious risks of efficiency decrease.

Towards $\mathbf{RQ_1}$ and $\mathbf{RQ_2}$, reduced ordered consistent PDDs can close the well-known gap between bbBDDs and DTs for concise control policy representation.

5.1.2 Comparison With Bit-blasted BDDs. For representing control policies, reduced ordered consistent PDDs can be expected to be more explainable than bbBDDs: First, BDDs need multiple Boolean variables to represent state variables, which makes the representation less comprehensible. For instance, if there is a simple predicate $temp \leq 19$, then already at least five decisions on five Boolean variables (required to numerically represent 19 through bit-blasting) have to be made. A single predicate used in a PDD cannot only express the conditions on state variables, but also often captures the relation between multiple state variables easily. An example for this can be found in the supplemental material [20].

Second, contributing to explainability by Occam's Razor, PDDs can be significantly more concise compared to bbBDDs, as we show in Table 1. Here, PDDs are on (geometric) average 77% smaller than bbBDDs. This is an improvement over DTs, which are 73% smaller than bbBDDs. In Figure 5, we provide an overview by comparing two normalized ratios: (i) *bbBDD size ratio* (ratio of the size of the constructed bbBDDs to the number of states in the system); and (ii) *PDD size ratio* (ratio of the size of the constructed PDDs to the number of states in the system). With the exception of two cases (*ij.10* and *pnueli-zuck.5*), the PDDs are notably smaller than the bbBDDs. In some cases (*blocksworld.5* and *cdrive.10*), the number of decision nodes in the bbBDDs are three times the size of the policy tables, not offering a smaller representation of a policy. In contrast, PDDs consistently succeed in providing compact representations.

For $\mathbf{RQ_1}$, reduced ordered consistent PDDs provide on average a more concise representation of control policies than bbBDDs.

5.1.3 Comparison with DTs. PDDs use predicates as in DTs, rendering decisions interpretable. However, due to merging isomorphic subdiagrams, common decision making can also be revealed through very same decision nodes, adding a component of explainability in contrast to DTs. A similar improvement in explainability can also be observed in software engineering when avoiding code duplication [29, 70]. Further, node sharing often decreases the size of the diagram, adding explainability following Occam's Razor. Thus, to assess the effectiveness of PDDs for explainability, we need to consider the effect of sharing of nodes due the subgraph merging.

Table 1: Sizes of controller representations: explicit states, as bit-blasted BDD, learnt DT, and in each step of the PDD pipeline. The upper part lists permissive policies from Scots and UppAal, while the lower part lists deterministic policies from Storm. The smallest value in each row is written in bold. PDD explainability is evaluated w.r.t. final PDDs after care-set reduction.

Controllers	Classical Representations			PDD Pipeline				PDD Explainability			
	States	bbBDD	DT	Plain	Reordered	Consistent	Care-set	#shared	$\frac{DT}{bbBDD}$	$\frac{PDD}{bbBDD}$	$\frac{PDD}{DT}$
10rooms	26244	1102	8648	1332	419	344	**344**	211	7.85	0.31	0.04
cartpole	271	197	**126**	206	172	133	**126**	0	0.64	0.64	1
cruise-latest	295615	2115	493	1091	554	479	**476**	66	0.23	0.23	0.97
dcdc	593089	814	**135**	149	135	135	**135**	0	0.17	0.17	1
helicopter	280539	3348	3169	10294	5577	3276	**3158**	486	0.95	0.94	1
traffic_30m	16639662	4522	6286	11088	5461	2497	**2350**	1538	1.39	0.52	0.37
beb.3-4.LineSeized	4173	1051	**32**	33	32	32	**32**	0	0.03	0.03	1
blocksworld.5	1124	4043	**617**	2646	1742	1526	796	13	0.15	0.20	1.29
cdrive.10	1921	6151	**1200**	9442	3828	3828	**1200**	0	0.20	0.20	1
consensus.2.disagree	2064	112	33	48	36	33	**31**	1	0.29	0.28	0.94
csma.2-4.some_before	7472	1172	**51**	78	60	58	53	2	0.04	0.05	1.04
eajs.2.100.5.ExpUtil	12627	1349	**83**	108	81	81	85	7	0.06	0.06	1.02
echoring.MaxOffline1	104892	48765	**934**	4429	1288	1274	970	94	0.02	0.02	1.04
elevators.a-11-9	14742	6790	8163	16563	6126	6077	**5555**	2756	1.20	0.82	0.68
exploding-blocksworld.5	76741	39436	**2490**	14857	6648	6504	3635	1346	0.06	0.09	1.46
firewire_abst.3.rounds	610	51	**12**	12	12	12	**12**	0	0.24	0.24	1
ij.10	1013	**415**	645	458	452	452	453	222	1.55	1.09	0.70
pacman.5	232	440	**21**	28	23	23	**21**	0	0.05	0.05	1
philosophers-mdp.3	344	251	**195**	310	246	205	203	5	0.78	0.81	1.04
pnueli-zuck.5	303427	**59217**	85685	1304402	496033	73139	72192	26941	1.45	1.22	0.84
rabin.3	704	301	**55**	80	62	59	58	0	0.18	0.19	1.05
rectangle-tireworld.11	241	259	**240**	281	274	240	**240**	0	0.93	0.93	1
triangle-tireworld.9	48	38	**13**	14	14	13	**13**	0	0.34	0.34	1
wlan_dl.0.80.deadline	189641	6591	**1684**	13763	3513	2004	1824	199	0.26	0.28	1.08
zeroconf.1000.4.true	1068	520	**41**	60	55	56	44	1	0.08	0.08	1.07
							Geometric means		0.27	0.23	0.84

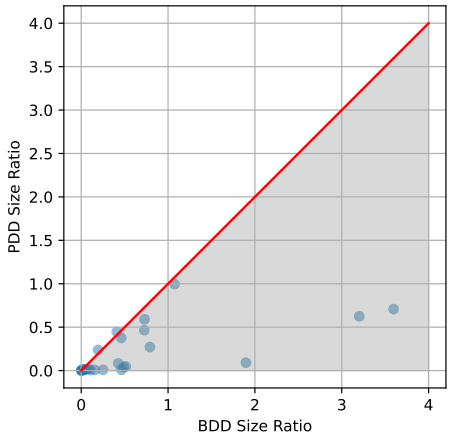


Figure 5: Comparison of normalized sizes of constructed PDDs and bbBDDs. Ratios of PDD (bbBDD, respectively) sizes to the number of states in the represented controller.

In terms of size, we examine whether reduction rules (such as subgraph merging) compensate for the strict order of predicates. As reported in Table 1, PDDs are on (geometric) average 16% smaller than the DTs. In Figure 6, we provide an overview by comparing two normalized ratios: (i) *DT size ratio* (ratio of the size of the constructed DTs to the number of states in the system); and (ii) *PDD size ratio* (ratio of the size of the constructed PDDs to the number of states in the system). In 8 of the examples, the generated PDDs have the same size as the DTs. For some large controllers, we have produced smaller PDDs in comparison to the DT representation.

Is the case of *10rooms*, where we compressed the DT by a factor of 25 while producing the PDD. Interestingly, the cases where bbBDDs provide smaller representation than DTs show that PDDs have either created the smallest representation among the three (in case of *10rooms*, *traffic_30m*, *elevators.a-11-9*) or managed to reduce the gap of size (in case of *ij.10*, *pnueli-zuck.5*).

The column marked with "#shared" in Table 1 gives the number of *shared nodes* in PDDs. In two cases (*10rooms* and *traffic_30m*), the PDDs contain more than 60% shared nodes (nodes with at least two parent edges). Note that, the DT-learning algorithms are optimized

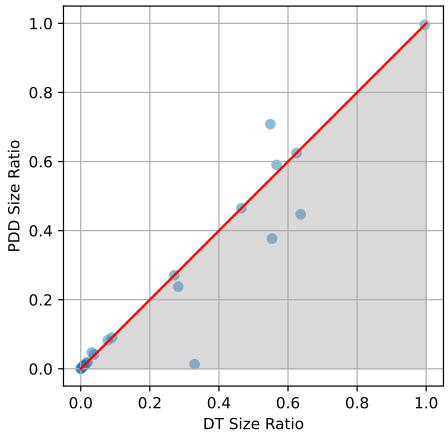


Figure 6: Comparison of normalized sizes of PDDs and DTs. For a structure $\mathcal{S} \in \{DT, PDD\}$, the $\mathcal{S}$ size ratio is the ratio of the size of $\mathcal{S}$ to the number of states $|S|$ in the system.

to create small tree-like structures without node sharing. As a result, only in larger models, we encounter isomorphic subdiagrams which can be merged while creating shared nodes in PDDs.

To illustrate this, we used the Israeli Jalfon randomized self-stabilizing protocol [42] from the Prism benchmarks. We extracted policies for different number of processes (3 to 17) and created PDD representations of the policies and observed the ratio of shared nodes in the PDD. With increasing number of processes, the number of states in the system grows exponentially. Smaller models have no shared nodes, but as the number of processes increases, the percentage of shared nodes in the PDD increases as well (See Figure 7; for a detailed description, see the supplemental material [20]). For example, with 17 processes, out of 34572 nodes in the PDD, more than 80% nodes have at least two parent edges. This demonstrates that PDDs become more efficient in larger models.

Answering $\mathbf{RQ_2}$, PDDs provide on average a more concise representation of control policies than DTs, largely benefitting from node sharing and common decision making.

5.2 Ablation Studies

Table 1 shows the impact of each step of our PDD synthesis pipeline (see Figure 4). Towards a fair comparison with DTs, we only counted the number of decision nodes and excluded duplicated action nodes that would massively add more nodes to DTs compared to DDs. We see that imposing an order on predicates usually increases the number of nodes and does not outweigh the possibility of merging isomorphic subdiagrams. Most drastically, this can be seen with *pnueli-zuck.5*, where the resulting PDD size is more than 15 times as big as the DT. Reordering then has great impact, reducing the sizes of the PDDs significantly, especially for the larger examples. For *pnueli-zuck.5*, it more than halves the number of nodes. As apparent from Table 1, consistency and care-set reduction have not always big impact but can also reduce PDD sizes: ensuring consistency can lead to 85% reduction (*pnueli-zuck.5*) and care-set reduction can reduce diagram sizes up to 68% (*cdrive.10*).

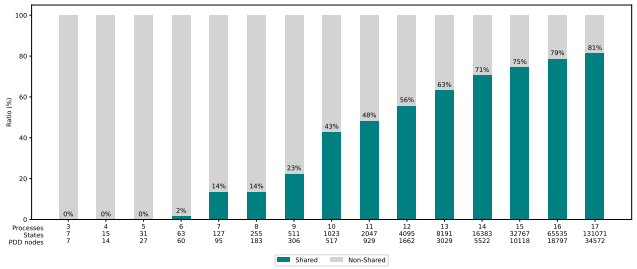


Figure 7: Fraction of shared nodes in PDDs for a self-stabilizing protocol with increasing number of processes.

Towards $\mathbf{RQ_3}$, we conclude that consistency checking, reordering, and care-set reductions are all effective for reducing PDD sizes and thus improve control strategy explanation.

6 Conclusion

Binary decision diagrams (BDDs) provide concise representations of data for which mature theory and broad tool support is available. Differently, decision trees (DTs) are well-known for their interpretability and enjoy many performant learning algorithms. Due to their predicates, DTs render also more explainable than BDDs with classical bit-blasting non-binary variables into bit-vectors. We introduced *predicate decision diagrams (PDDs)* along with a synthesis pipeline to generate PDDs from DTs and exploit BDD reduction techniques towards concise representations. With PDDs, we established a representation for compact control policies that unite the benefits of both data structures, BDDs and DTs. We found that the sizes of PDDs are on par with DT in most of the control policies we investigated. This improves the state of the art in BDD-based representations such that PDDs can now be used as a viable alternative to DT as explainable data structure, but with also indicating common decision making through subdiagram merging.

For future work, an integration into state-of-the-art BDD packages [41] to directly support predicates and extend linear DDs [19] would improve applicability. Further, adaptions of approximation algorithms on PDDs [30, 58] could extract the essence of control strategies in even smaller and thus more explainable PDDs. It would be also interesting to establish theoretical guarantees on PDD sizes [62] or regarding different explainability metrics to underpin the benefits of PDDs. We further envision a PDD learning algorithm that specifically prefers choices of predicates to increase sharing and could directly replace DT learning algorithms and benefit from the advantages of PDDs we showed in this paper.

Acknowledgments

Authors in alphabetic order. This work was partially supported by the DFG under the projects TRR 248 (see https://perspicuous-computing.science, project ID 389792660) and EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), by the NWO through Veni grant VI.Veni.222.431, and the MUNI Award in Science and Humanities (MUNI/I/1757/2021) of the Grant Agency of Masaryk University.

References

[1] 2019. Ethics Guidelines for Trustworthy AI - European Commission, Directorate-General for Communications Networks, Content and Technology. https://data.europa.eu/doi/10.2759/177365.

[2] 2020. Four Principles of Explainable Artificial Intelligence - (U.S.) National Institute of Standards and Technology (NIST). https://doi.org/10.6028/NIST.IR.8312-draft.

[3] 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.

[4] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access* 6 (2018), 52138–52160.

[5] Pranav Ashok, Tomás Brázdil, Krishnendu Chatterjee, Jan Křetínský, Christoph H. Lampert, and Viktor Toman. 2019. Strategy Representation by Decision Trees with Linear Classifiers. In *QEST (Lecture Notes in Computer Science, Vol. 11785)*. Springer, 109–128.

[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 '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, Aaron D. Ames, Sanjit A. Seshia, and Jyotirmoy Deshmukh (Eds.). ACM, 30:1–30:2. https://doi.org/10.1145/3365365.3383468

[7] Pranav Ashok, Mathias Jackermeier, Jan Kretínský, 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] Pranav Ashok, Jan Křetínský, Kim Guldstrand Larsen, Adrien Le Coënt, Jakob Haahr Taankvist, and Maximilian Weininger. 2019. SOS: Safe, Optimal and Small Strategies for Hybrid Markov Decision Processes. In *QEST (Lecture Notes in Computer Science, Vol. 11785)*. Springer, 147–164.

[9] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. 1997. Algebraic Decision Diagrams and Their Applications. *Formal Methods in System Design* 10, 2 (1997), 171–206.

[10] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. 2021. Satisfiability Modulo Theories. In *Handbook of Satisfiability, Second Edition*, Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press, Chapter 33, 825–885. http://theory.stanford.edu/~barrett/pubs/BSST21.pdf

[11] B. Bollig and I. Wegener. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Comput.* 45, 9 (1996), 993–1002. https://doi.org/10.1109/12.537122

[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, Jan Křetínský, and Viktor Toman. 2018. Strategy Representation by Decision Trees in Reactive Synthesis. In *TACAS (1) (Lecture Notes in Computer Science, Vol. 10805)*. Springer, 385–407.

[15] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.

[16] Randal E. Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers* 35, 8 (1986), 677–691.

[17] Gianpiero Cabodi, Paolo E. Camurati, Alexey Ignatiev, João Marques-Silva, Marco Palena, and Paolo Pasini. 2021. Optimizing Binary Decision Diagrams for Interpretable Machine Learning Classification. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*. IEEE, 1122–1125. https://doi.org/10.23919/DATE51398.2021.9474083

[18] Roberto Cavada, Alessandro Cimatti, Anders Franzen, Krishnamani Kalyanasundaram, Marco Roveri, and R.K. Shyamasundar. 2007. Computing Predicate Abstractions by Integrating BDDs and SMT Solvers. In *Formal Methods in Computer Aided Design (FMCAD'07)*. 69–76. https://doi.org/10.1109/FAMCAD.2007.35

[19] Sagar Chaki, Arie Gurfinkel, and Ofer Strichman. 2009. Decision diagrams for linear arithmetic. In *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*. IEEE, 53–60. https://doi.org/10.1109/FMCAD.2009.5351143

[20] Debraj Chakraborty, Clemens Dubslaff, Sudeep Kanav, Jan Kretinsky, and Christoph Weinhuber. 2025. Explaining Control Policies through Predicate Decision Diagrams. arXiv:2503.06420 [cs.AI] https://arxiv.org/abs/2503.06420

[21] Kenil C. K. Cheng and Roland H. C. Yap. 2005. Constrained Decision Diagrams. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, Manuela M. Veloso and Subbarao Kambhampati (Eds.). AAAI Press / The MIT Press, 366–371. http://www.aaai.org/Library/AAAI/2005/aaai05-058.php

[22] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. 1998. Automatic OBDD-Based Generation of Universal Plans in Non-Deterministic Domains. In *AAAI/IAAI*. AAAI Press / The MIT Press, 875–881.

[23] O. Coudert and J.C. Madre. 1990. A unified framework for the formal verification of sequential circuits. In *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*. 126–129. https://doi.org/10.1109/ICCAD.1990.129859

[24] Adnan Darwiche. 2011. SDD: a new canonical representation of propositional knowledge bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two* (Barcelona, Catalonia, Spain) *(IJCAI'11)*. AAAI Press, 819–826.

[25] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taankvist. 2015. Uppaal Stratego. In *TACAS (Lecture Notes in Computer Science, Vol. 9035)*. Springer, 206–211.

[26] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. 2017. A Storm is Coming: A Modern Probabilistic Model Checker. In *CAV (2) (Lecture Notes in Computer Science, Vol. 10427)*. Springer, 592–600.

[27] Giuseppe Della Penna, Benedetto Intrigila, Nadia Lauri, and Daniele Magazzeni. 2009. Fast and Compact Encoding of Numerical Controllers Using OBDDs. In *Informatics in Control, Automation and Robotics: Selcted Papers from the International Conference on Informatics in Control, Automation and Robotics 2008*, Juan Andrade Cetto, Jean-Louis Ferrier, and Joaquim Filipe (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 75–87.

[28] Clemens Dubslaff, Nils Husung, and Nikolai Käfer. 2024. Configuring BDD Compilation Techniques for Feature Models. In *Proceedings of the 28th ACM International Systems and Software Product Line Conference* (Dommeldange, Luxembourg) *(SPLC '24)*. Association for Computing Machinery, New York, NY, USA, 209–216. https://doi.org/10.1145/3646548.3676538

[29] Clemens Dubslaff, Verena Klös, and Juliane Päßler. 2024. Template Decision Diagrams for Meta Control and Explainability. In *Explainable Artificial Intelligence*, Luca Longo, Sebastian Lapuschkin, and Christin Seifert (Eds.). Springer Nature Switzerland, Cham, 219–242.

[30] Clemens Dubslaff and Joshua Wirtz. 2025. *Compiling Binary Decision Diagrams with Interrupt-Based Downsizing*. Springer Nature Switzerland, Cham, 252–273. https://doi.org/10.1007/978-3-031-75778-5_12

[31] Alexandre M. Florio, Pedro Martins, Maximilian Schiffer, Thiago Serra, and Thibaut Vidal. 2023. Optimal Decision Diagrams for Classification. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 6 (Jun. 2023), 7577–7585. https://doi.org/10.1609/aaai.v37i6.25920

[32] Pascal Fontaine and E. Pascal Gribomont. 2002. Using BDDs with Combinations of Theories. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Matthias Baaz and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 190–201.

[33] Jan Friso Groote and Jaco van de Pol. 2000. Equational Binary Decision Diagrams. In *Logic for Programming and Automated Reasoning*, Michel Parigot and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 161–178.

[34] Masahiro Fujita, Patrick C. McGeer, and Jerry Chih-Yuan Yang. 1997. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. *Formal Methods Syst. Des.* 10, 2/3 (1997), 149–169.

[35] Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth. 2016. Learning invariants using decision trees and implication counterexamples. In *POPL*. ACM, 499–512.

[36] I. J. Good. 1977. Explicativity: a mathematical theory of explanation with statistical applications. *Proc. R. Soc. Lond.* A354 (1977), 303–330.

[37] Ujjwal Das Gupta, Erik Talvitie, and Michael Bowling. 2015. Policy tree: Adaptive representation for policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[38] Arnd Hartmanns, Michaela Klauck, David Parker, Tim Quatmann, and Enno Ruijters. 2019. The Quantitative Verification Benchmark Set. In *TACAS (1) (Lecture Notes in Computer Science, Vol. 11427)*. Springer, 344–350.

[39] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. 1999. SPUDD: Stochastic Planning using Decision Diagrams. In *UAI*. Morgan Kaufmann, 279–288.

[40] Hao Hu, Marie-José Huguet, and Mohamed Siala. 2022. Optimizing Binary Decision Diagrams with MaxSAT for Classification. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 3767–3775. https://doi.org/10.1609/AAAI.V36I4.20291

[41] Nils Husung, Clemens Dubslaff, Holger Hermanns, and Maximilian A. Köhl. 2024. OxiDD. In *Tools and Algorithms for the Construction and Analysis of Systems*, Bernd Finkbeiner and Laura Kovács (Eds.). Springer Nature Switzerland, Cham, 255–275.

[42] Amos Israeli and Marc Jalfon. 1990. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing* (Quebec City, Quebec, Canada) *(PODC '90)*. Association for Computing Machinery, New York, NY, USA, 119–131. https://doi.org/10.1145/93385.93409

[43] Manuel Mazo Jr., Anna Davitian, and Paulo Tabuada. 2010. PESSOA: A Tool for Embedded Controller Synthesis. In *CAV (Lecture Notes in Computer Science, Vol. 6174)*. Springer, 566–569.

[44] Florian Jüngermann, Jan Kretínský, and Maximilian Weininger. 2023. Algebraically explainable controllers: decision trees and support vector machines join forces. *Int. J. Softw. Tools Technol. Transf.* 25, 3 (2023), 249–266.

[45] M. Kwiatkowska, G. Norman, and D. Parker. 2012. The PRISM Benchmark Suite. In *Proc. 9th International Conference on Quantitative Evaluation of SysTems (QEST'12)*. IEEE CS Press, 203–204.

[46] Jørn Lind-Nielsen. 2004. BuDDy: A binary decision diagram package, version 2.4. https://buddy.sourceforge.net/manual/

[47] Kenneth L. McMillan. 1993. *Symbolic Model Checking*. Springer US, Boston, MA.

[48] D.M. Miller. 1993. Multiple-valued logic design tools. In *[1993] Proceedings of the Twenty-Third International Symposium on Multiple-Valued Logic*. 2–11. https://doi.org/10.1109/ISMVL.1993.289589

[49] Shin-ichi Minato. 1996. *Binary decision diagrams and applications for VLSI CAD*. Kluwer Academic Publishers, USA.

[50] T. M. Mitchell. 1997. *Machine learning*. McGraw-Hill.

[51] Luis E. Gonzalez Moctezuma, Andrei Lobov, and Jose L. Martinez Lastra. 2012. Decision making by using tree-like structures on industrial controllers. In *2012 Tenth International Conference on ICT and Knowledge Engineering*. 77–83. https://doi.org/10.1109/ICTKE.2012.6408575

[52] Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. 1999. Difference Decision Diagrams. In *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1683)*, Jörg Flum and Mario Rodríguez-Artalejo (Eds.). Springer, 111–125. https://doi.org/10.1007/3-540-48168-0_9

[53] Shinobu Nagayama, Tsutomu Sasao, and Jon T. Butler. 2007. Numerical Function Generators Using Edge-Valued Binary Decision Diagrams. In *Proceedings of the 12th Conference on Asia South Pacific Design Automation, ASP-DAC 2007, Yokohama, Japan, January 23-26, 2007*. IEEE Computer Society, 535–540. https://doi.org/10.1109/ASPDAC.2007.358041

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

[55] Larry D Pyeatt, Adele E Howe, et al. 2001. Decision tree function approximation in reinforcement learning. In *Proceedings of the third international symposium on adaptive systems: evolutionary computation and probabilistic graphical models*, Vol. 2. Cuba, 70–77.

[56] J. Ross Quinlan. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (1986), 81–106.

[57] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

[58] Kavita Ravi, Kenneth L. McMillan, Thomas R. Shiple, and Fabio Somenzi. 1998. Approximation and decomposition of binary decision diagrams. In *Proceedings of the 35th annual Design Automation Conference* (San Francisco, California, USA) *(DAC '98)*. Association for Computing Machinery, New York, NY, USA, 445–450. https://doi.org/10.1145/277044.277168

[59] Michael Rice and Sanjay Kulhari. 2008. A survey of static variable ordering heuristics for efficient BDD/MDD construction. *University of California, Tech. Rep* (2008), 130.

[60] R. Rudell. 1993. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. of the IEEE/ACM Conference on Computer-Aided Design (ICCAD)*. IEEE Computer Society, 42–47.

[61] Matthias Rungger and Majid Zamani. 2016. SCOTS: A Tool for the Synthesis of Symbolic Controllers. In *HSCC*. ACM, 99–104.

[62] Steffan Christ Sølvsten and Jaco van de Pol. 2023. Predicting Memory Demands of BDD Operations Using Maximum Graph Cuts. In *Automated Technology for Verification and Analysis*, Étienne André and Jun Sun (Eds.). Springer Nature Switzerland, Cham, 72–92.

[63] Robert St-Aubin, Jesse Hoey, and Craig Boutilier. 2000. APRICODD: Approximate Policy Construction Using Decision Diagrams. In *NIPS*. MIT Press, 1089–1095.

[64] Cesare Tinelli and Mehdi Harandi. 1996. A New Correctness Proof of the Nelson-Oppen Combination Procedure. In *Frontiers of Combining Systems: First International Workshop, Munich, March 1996*, Frans Baader and Klaus U. Schulz (Eds.). Springer Netherlands, Dordrecht, 103–119.

[65] Hazem Torfah, Shetal Shah, Supratik Chakraborty, S Akshay, and Sanjit A Seshia. 2021. Synthesizing pareto-optimal interpretations for black-box models. In *Formal Methods in Computer Aided Design*. IEEE.

[66] Hélene Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. 2020. Learning optimal decision trees using constraint programming. *Constraints* 25 (2020), 226–250.

[67] Sicco Verwer and Yingqian Zhang. 2019. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 1625–1632.

[68] Daniël Vos and Sicco Verwer. 2023. Optimal decision tree policies for Markov decision processes. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 5457–5465.

[69] Ivan S. Zapreev, Cees Verdier, and Manuel Mazo Jr. 2018. Optimal Symbolic Controllers Determinization for BDD storage. In *ADHS 2018 (IFAC-PapersOnLine, Vol. 51)*. Elsevier, 1–6. https://doi.org/10.1016/j.ifacol.2018.08.001

[70] Horst Zuse. 2019. *Software complexity: measures and methods*. Vol. 4. Walter de Gruyter GmbH & Co KG.