Symbolic Functional Decomposition: A Reconfiguration Approach

Mateus de Oliveira Oliveira[1,2], **Wim Van den Broeck**[2]

[1]Department of Computer and Systems Sciences, Stockholm University, Sweden
[2]Department of Informatics, University of Bergen, Norway
oliveira@dsv.su.se, wim.broeck@uib.no

Abstract

Functional decomposition is the process of breaking down a function f into a composition $f = g(f_1, \ldots, f_k)$ of simpler functions $f_1, \ldots, f_k$ belonging to some class $\mathcal{F}$. This fundamental notion can be used to model applications arising in a wide variety of contexts, ranging from machine learning to formal language theory. In this work, we study functional decomposition by leveraging on the notion of functional reconfiguration. In this setting, constraints are imposed not only on the factor functions $f_1, \ldots, f_k$ but also on the intermediate functions arising during the composition process.

We introduce a symbolic framework to address functional reconfiguration and decomposition problems. In our framework, functions arising during the reconfiguration process are represented symbolically, using ordered binary decision diagrams (OBDDs). The function g used to specify the reconfiguration process is represented by a Boolean circuit C. Finally, the function class $\mathcal{F}$ is represented by a second-order finite automaton $\mathcal{A}$. Our main result states that functional reconfiguration, and hence functional decomposition, can be solved in fixed-parameter linear time when parameterized by the width of the input OBDD, by structural parameters associated with the reconfiguration circuit C, and by the size of the second-order finite automaton $\mathcal{A}$.

1 Introduction

Cognition and learning are processes that often occur in a hierarchical manner, where complex concepts are broken down into simpler, more manageable ones (Goel and Chandrasekaran 1989; Correa et al. 2023; Atzmon et al. 2020; Wang et al. 2023). In contexts where concepts can be represented by functions, this hierarchical approach to processing and understanding information can often be formalized using an appropriate notion of functional decomposition (Zupan et al. 1997; Bohanec and Zupan 2004; Vykhovanets 2006; Deniziak and Wiśniewski 2021).

In this work, we view functional decomposition as the process of breaking down a Boolean function f into a composition $g(f_1, \ldots, f_k)$, where $f_1, \ldots, f_k$ are functions from a pre-defined class $\mathcal{F}$. The function g specifies how the component functions $f_1, \ldots, f_k$ combine to form the target function f, while $\mathcal{F}$ is a class of functions that are considered to be simpler than the target function f. For instance, $\mathcal{F}$ may be a class of functions whose preimage is closed under some algebraic operation, or a class of functions of bounded complexity with respect to some suitable complexity measure.

Functional decomposition is a fundamental concept within the realm of artificial intelligence because it promotes modularity, interpretability and explainability. First, functional decomposition enables the reuse of well-defined components across distinct AI systems (Sun, Nguyen, and Mamitsuka 2019; Izza et al. 2020; Srivastava 2023). In this context one could hope to implement a system with a given functionality f by combining pre-existing systems with functionality $f_1, f_2, \ldots, f_k$. Second, decomposing a complex function into sub-functions makes the model more interpretable. Each sub-function can be understood as performing a specific task, or transformation, which provides insights on how the model arrives at its final computations. Finally, it helps in explaining the decision-making process of computational models by breaking down the decision into simpler, easier to understand steps (Molnar, Casalicchio, and Bischl 2020; Zupan et al. 1999; Hashemi and Vikalo 2018; Hiabu, Meyer, and Wright 2023; Laberge et al. 2024).

As a closely related concept, in this work we also consider the notion of *functional reconfiguration*. In this setting, not only the factor functions $f_1, \ldots, f_k$ are required to satisfy some property, but also the sub-functions arising during the composition process. Intuitively, functional reconfiguration maintains consistency throughout the decomposition process. This consistency contributes even further to interpretability and explainability because it ensures that each part of the model adheres to a uniform structure or set of principles.

We analyze both functional decomposition and functional reconfiguration through the lenses of parameterized complexity theory. In our framework, functions are represented symbolically, using ordered binary decision diagrams (OBDDs), a prominent formalism for the symbolic representation of Boolean functions (Bryant 1992). The reconfiguration process is described using a Boolean circuit C. Finally, the function class $\mathcal{F}$ is specified using a second-order finite automaton $\mathcal{A}$, an automata-theoretic formalism for the representation of classes of functions. Our main results state that functional reconfiguration, and functional decomposition can be solved in fixed-parameter linear time when pa-

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

rameterized by the number of factor functions, the width of the OBDDs representing these functions, the size of the circuit C and the size of the second-order automaton $\mathcal{A}$.

1.1 Our Results

For a fixed Boolean circuit C with k inputs and a fixed class of functions $\mathcal{F}$, we consider the problem of decomposing a given Boolean function $f : \{0,1\}^n \to \{0,1\}$ as a C-combination of functions $f_1, \ldots, f_k : \{0,1\}^n \to \{0,1\}$ from $\mathcal{F}$. More specifically, we require that for each $x \in \{0,1\}^n$, $f(x) = F_C(f_1(x), \ldots, f_k(x))$, where F_C is the k-bit function computed by C. We say that the sequence $f_1, \ldots, f_k$ is a $(C, \mathcal{F})$-decomposition of f.

In order for the $(C, \mathcal{F})$-decomposition problem to make sense from a computational point of view, we also need to specify a model for the representation of the input function f and factor functions $f_1, \ldots, f_k$, as well as a model for the representation of the class of functions $\mathcal{F}$. In our framework, we represent Boolean functions symbolically, using ordered binary decision diagrams (OBDDs), which are one of the most prominent formalisms for symbolic computation (Bryant 1992). An OBDD D is, essentially, an acyclic deterministic finite automaton where the set of states is partitioned into a sequence of levels $X_0X_1...X_n$ and the set of transitions is partitioned into a sequence of layers $B_1B_2 \ldots B_n$, where each layer B_i contains only transitions from states in X_{i-1} to X_i. Such an OBDD D represents a function $f_D : \{0,1\}^n \to \{0,1\}$ where for each $x \in \{0,1\}^n$, $f(x) = 1$ if and only if D accepts x. At the same time that OBDDs may provide a much more concise representation of a Boolean function when compared to representations based on truth tables or decision trees, OBDDs enjoy a series of algorithmic properties, such as efficient computation of union, intersection, complementation, etc (Bryant 1992).

An important complexity measure in the context of symbolic computation is the *width* of an OBDD, that is to say, the maximum number of states in one of its levels (Bollig 2014, 2012; Hachtel and Somenzi 1993; Woelfel 2006; Sawitzki 2004). For each fixed $p \in \mathbb{N}$, an OBDD of width at most p can be encoded as a word over the alphabet $\widehat{\mathcal{B}}(p)$ of all layers of width at most p. This allows one to define classes of Boolean functions using automata theoretic formalisms. More specifically, a finite automaton $\mathcal{A}$ over $\widehat{\mathcal{B}}(p)$ defines the class $\mathcal{F}(\mathcal{A}) = \{f_D : D \in \mathcal{L}(\mathcal{A})\}$ of all functions computed by OBDDs accepted by $\mathcal{A}$. A crucial feature of this approach is the fact that, as shown in (de Melo and de Oliveira Oliveira 2022), second-order finite automata can be canonized with respect to the *class of functions* it represents, and classes of functions represented by second-order automata are closed under usual Boolean operations, as well as several other higher-order operations (de Melo and de Oliveira Oliveira 2022).

In this work, we show that the $(C, \mathcal{F}(\mathcal{A}))$-decomposition problem can be solved in fixed-parameter tractable linear time when parameterized by the number k of factor functions, the width p of the OBDDs representing the factor functions $f_1, \ldots, f_k$, the number m of gates in the circuit C, and the size $|\mathcal{A}|$ of the second-order finite automaton representing the class $\mathcal{F}$.

Problem name: FUNCTIONALDECOMPOSITION
Given: Numbers $k, p, m \in \mathbb{N}$, an OBDD D, a Boolean circuit C with k inputs and m gates, and a second-order finite automaton $\mathcal{A}$ over $\widehat{\mathcal{B}}(p)$.
Parameters: k, p, m, $|\mathcal{A}|$.
Question: Are there OBDDs $D_1, \ldots, D_k \in \mathcal{F}(\mathcal{A})$ of width at most p such that $f_D = F_C(f_{D_1}, \ldots, f_{D_k})$?

Our main result (Theorem 10) states that FUNCTIONALDECOMPOSITION can be solved in time

$$2^{p^{O(2^m)}} \cdot |\mathcal{A}|^k \cdot |D|$$

where $|D|$ is the size of the input OBDD representing the function to be decomposed. We note that when the parameters k, p, m, $|\mathcal{A}|$ are fixed, our algorithm works in linear time on the size of the input OBDD D.

Let us give a brief motivation for our parameterization. First, we note that if an OBDD of length n has width p then the number of bits necessary to specify the OBDD is $O(p \cdot n)$. In usual decomposition problems, the parameter k is often set to 2, since the goal in this case is to factorize a given mathematical object into two simpler mathematical objects. Any Boolean function on k inputs can be computed by a Boolean circuit with at most $O(2^k/k)$ gates. Therefore the parameter m is upper bounded by a function of k. Nevertheless, it is also reasonable to have the size of the circuit as a parameter because this size can be much smaller than the trivial upper bound. For instance, the AND function on k bits can be computed by a circuit with $k-1$ gates (of fan-in 2). Finally, many interesting classes of functions, can be represented by second-order finite automata of constant size (Kuske 2021; de Melo and de Oliveira Oliveira 2022).

We prove our main result within the framework of functional reconfiguration. More specifically, we introduce the notion of *reconfiguration width* of a circuit C with respect to input functions $f_1, \ldots, f_k$ (Definition 7), a complexity measure that may be of independent interest. Intuitively, this measure captures the maximum complexity of a function arising during the process of reconfiguring the input functions $f_1, \ldots, f_k$ into the output function $f = F_C(f_1, \ldots, f_k)$ according to the specification provided by circuit C. In Section 4, we introduce the problem FUNCTIONALRECONFIGURATION, a reconfiguration variant of FUNCTIONALDECOMPOSITION where the reconfiguration width is requested to be below a threshold w. We show that this problem can be solved in time $2^{\mathcal{O}(m \cdot w^2 \log w)} \cdot |\mathcal{A}|^k \cdot |D|$ (Theorem 8). Our main result follows by combining Theorem 8 with an upper bound on the reconfiguration width in terms of the width of the factor functions and of the size of the circuit (Lemma 9).

2 Preliminaries

2.1 Basics

We let $\mathbb{N} = \{0, 1, \ldots\}$ be the set of *natural numbers*, and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ be the set of positive natural numbers.

For each $n \in \mathbb{N}$ we set $[n] = \{1, 2, \ldots, n\}$. According to this notation, $[0] = \emptyset$. For each $w \in \mathbb{N}_+$, we let $[\![w]\!] = \{0, 1, \ldots, w-1\}$.

An *alphabet* is any finite, non-empty set Σ. A *string* over Σ is a finite sequence of symbols from Σ. We let Σ^* be the set of all strings over Σ, including the empty string λ, and let $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ be the set of non-empty strings over Σ. A *language* over Σ is any subset $L \subseteq \Sigma^*$. For each $k \in \mathbb{N}_+$, we let Σ^k be the set of all strings of length k over Σ.

2.2 α-Regular Languages

The proofs of our results heavily rely on techniques from automata theory. For this reason, we briefly recall some basic concepts about finite automata and regular languages.

A *finite automaton* is a tuple $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ where Σ is an alphabet, Q is a finite set of *states*, $I \subseteq Q$ is a set of *initial states*, $F \subseteq Q$ is a set of *final states*, and $\Delta \subseteq Q \times \Sigma \times Q$ is a set of *transitions*. A string $x \in \Sigma^n$ is said to be *accepted* by $\mathcal{A}$ if there is a sequence

$$(q_0, x_1, q_1)(q_1, x_2, q_2) \ldots (q_{n-1}, x_n, q_n)$$

such that $q_0 \in I$, $q_n \in F$, and for each $i \in [n]$, (q_{i-1}, x_i, q_i) is a transition in Δ. The *language* of $\mathcal{A}$ is defined as the set

$$\mathcal{L}(\mathcal{A}) = \{x \mid x \text{ is accepted by } \mathcal{A}\}$$

of all strings accepted by $\mathcal{A}$. We note that finite automata considered in this work may be non-deterministic, meaning that for some state q and some symbol $a \in \Sigma$, we may have two states q' and q'' such that (q, a, q') and (q, a, q'') are transitions in $\mathcal{A}$.

Definition 1. *Let $\alpha \in \mathbb{N}$ and $L \subseteq \Sigma^*$. We say that L is α-regular if there is a finite automaton $\mathcal{A}$ over Σ with at most α states such that $\mathcal{L}(\mathcal{A}) = L$.*

In the product construction for finite automata, two automata $\mathcal{A}$ and $\mathcal{A}'$ are combined into a single automaton $\mathcal{A} \times \mathcal{A}'$ whose states are pairs of states (q, q') from $\mathcal{A}$ and $\mathcal{A}'$, respectively, with transitions defined such that $(q, q') \xrightarrow{\sigma} (r, r')$ if and only if $q \xrightarrow{a} r$ in A and $q' \xrightarrow{a} r'$ in $\mathcal{A}'$. Initial (final) states are pairs of initial (final) states of the respective automata. It can be shown that $L(\mathcal{A} \times \mathcal{A}') = L(\mathcal{A}) \cap L(\mathcal{A}')$ (Hopcroft 1971). The following observation is a direct consequence of this construction.

Observation 2. *Let L be an α-regular language, and L' be an α'-regular language. Then $L \cap L'$ is $(\alpha \cdot \alpha')$-regular.*

2.3 Boolean Circuits

We view a Boolean circuit with input variables $\{x_1, \ldots, x_k\}$ as a sequence $C = (g_1, g_2, \ldots, g_m)$ of gates satisfying the following conditions:

1. Each gate g_i is of one of the following types:
   (a) a variable in the set $\{x_1, ..., x_k\}$;
   (b) a NOT gate $g_i = \texttt{NOT}(g_j)$ for some $j < i$;
   (c) an AND gate $g_i = \texttt{AND}(g_j, g_l)$ for some $j, l < i$;
   (d) an OR gate $g_i = \texttt{OR}(g_j, g_l)$ for some $j, l < i$.
2. For each $i \in [k]$, there is exactly one $j \in [m]$ such that $g_j = x_i$.
3. g_m is the only gate that is an output gate, i.e., a gate that is not an input to any other gate.

Given a Boolean circuit $C = (g_1, g_2, \ldots, g_m)$ and an index $i \in [m]$, we let $F_{g_i} : \{0,1\}^k \to \{0,1\}$ denote the Boolean function computed at gate g_i. We may write F_C to denote the Boolean function F_{g_m} computed by the output gate of C. We say that a function $f : \{0,1\}^n \to \{0,1\}$ is a Boolean combination of functions $f_1, \ldots, f_k : \{0,1\}^n \to \{0,1\}$ if there is a Boolean circuit C such that $f = F_C(f_1, \ldots, f_k)$. More specifically, for each $x \in \{0,1\}^n$, we have that $f(x) = F_C(f_1(x), \ldots, f_k(x))$.

Let $C = (g_1, \ldots, g_m)$ be a Boolean circuit. The depth of a gate g_i is inductively defined as follows. If g_i is an input gate, then $d(g_i) = 0$. On the other hand, if g_i is a non-input gate, then the depth of g_i is defined as the maximum depth of an input for g_i plus 1.

3 Automata vs Function Classes

In its most traditional setting, finite automata are used to represent sets of strings over a given finite alphabet Σ, or equivalently, as a way to represent functions of type $\Sigma^* \to \{0,1\}$. More recently, in a wide variety of contexts, automata theoretic formalisms have been defined with the intention to represent sets of sets of strings, or equivalently, function classes (Jain, Luo, and Stephan 2012; Case, Jain, and Stephan 2013; Zaid, Grädel, and Reinhardt 2017; Kuske 2021; de Melo and de Oliveira Oliveira 2022). In this section we describe the approach from (de Melo and de Oliveira Oliveira 2022), which is based on the notion of a second-order finite automaton. Our interest in this formalism stems from the fact that function classes represented in this way enjoy several useful closure and decidability properties, such as closure under intersection, union, bounded-width complementation, besides having decidable inclusion and emptiness of intersection tests.

3.1 Ordered Binary Decision Diagrams

We start by defining the notion of an ordered binary decision diagram using a slightly different notation than the one usually employed in the literature (Bryant 1992). The reason is that in our work, it will be more convenient to view an ordered binary decision diagram of width at most w as a sequence of symbols over an alphabet $\widehat{\mathcal{B}}(w)$ of w-layers.

Let $w \in \mathbb{N}_+$. We call a subset $B \subseteq [\![w]\!] \times \{0,1\} \times [\![w]\!]$ a *w-layer*. We let $\widehat{\mathcal{B}}(w)$ denote the set of all w-layers. For each $B \in \widehat{\mathcal{B}}(w)$, we let

$$\mathrm{Dom}(B) = \{q \mid \exists (a, q'), (q, a, q') \in B\}$$

be the *domain* of B, and

$$\mathrm{Im}(B) = \{q' \mid \exists (q, a), (q, a, q') \in B\}$$

be the *image* of B. Given $n, w \in \mathbb{N}_+$, an *ordered binary decision diagram* (OBDD) of length n and width at most w is a sequence $D = B_1 B_2 \ldots B_n$ of w-layers satisfying the following conditions:

1. $\mathrm{Dom}(B_1) = \{0\}$,
2. for each $i \in [n-1]$, $\mathrm{Im}(B_i) = \mathrm{Dom}(B_{i+1})$,

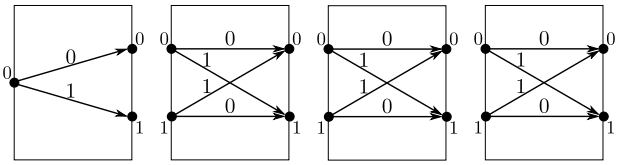

Figure 1: An OBDD of width 2 and length 4 accepting all strings in $\{0,1\}^4$ of odd parity. Note that each string of odd parity (say, 1011) reaches the element 1 in the image of the last layer, while each string of even parity (say, 1010) reaches the element 0 in the image of the last layer.

3. for each $i \in [n]$, each $q \in \mathrm{Dom}(B_i)$, and each $a \in \{0,1\}$, there is a unique q' such that $(q,a,q') \in B_i$.

We say that a string $x = x_1x_2\ldots x_n$ is accepted by D if there is a sequence

$$(q_0,x_1,q_1)(q_1,x_2,q_2)\ldots(q_{n-1},x_n,q_n)$$

where for each $i \in [n]$, $(q_{i-1},x_i,q_i) \in B_i$, and $q_n \in [\![w]\!] \setminus \{0\}$. Intuitively, x is accepted by D if it reaches a nonzero element in the image of the last layer of D. The language of D is the set

$$\mathcal{L}(D) = \{s \mid s \text{ is accepted by } D\}.$$

We may also regard an OBDD of length n as a representation of a binary function $f_D : \{0,1\}^n \to \{0,1\}$. More specifically, for each $s \in \{0,1\}^n$, we set $f_D(s) = 1$ if and only if $s \in \mathcal{L}(D)$.

In Figure 1, we depict an example of an OBDD. Each layer is represented by a box, while each triple (q,a,q') in such a layer is represented by an arrow $q \xrightarrow{a} q'$.

We say that an OBDD $D = B_1B_2,\ldots B_n$ is *normalized* if for each $i \in [n]$, and each two $j,j' \in \mathrm{Im}(B_i)$ with $j < j'$, the lexicographically-first string in $\{0,1\}^i$ that reaches j is smaller than the lexicographically-first string in $\{0,1\}^i$ that reaches j'. For instance, the OBDD D depicted in Figure 1 is normalized because for each $i \in [4]$, the lexicographically-first string that reaches the element 0 in the image of B_i is the string 0^i, while the lexicographically-first string that that reaches the element 1 is the string $0^{i-1}1$.

The size of an OBDD $D = B_1B_2\ldots B_n$, denoted by $|D|$ is defined as $|D| = |\mathrm{Dom}(B_1)| + \sum_{i\in[n]} |\mathrm{Im}(B_i)|$. The width of D is defined as $\mathrm{w}(D) = \max\{|\mathrm{Im}(B_i)|\}$. The next theorem, which is well known in the OBDD literature, states that each finite language $L \subseteq \{0,1\}^n$ corresponds to a unique normalized OBDD D of minimum size such that $\mathcal{L}(D) = L$. Additionally this OBDD has also minimum width.

Theorem 3 ((Bryant 1992; de Melo and de Oliveira Oliveira 2022))**.** *For each OBDD D there is a unique normalized OBDD $\mathcal{C}(D)$ of minimum size with the property that $\mathcal{L}(D) = \mathcal{L}(\mathcal{C}(D))$. Additionally, $\mathcal{C}(D)$ has minimum width among all OBDDs D' with $\mathcal{L}(D') = \mathcal{L}(D)$.*

We note that if D is an OBDD of length n and width at most w then $\mathcal{C}(D)$ can be constructed in time $O(w \cdot n)$, using the standard algorithm for minimization of acyclic finite automata, followed by normalization (Bryant 1992; de Melo and de Oliveira Oliveira 2022). The fact that $\mathcal{C}(D)$ is normalized guarantees that it is *syntactically unique*, meaning that for each two OBDDs D and D', $\mathcal{L}(D) = \mathcal{L}(D')$ if and only if $\mathcal{C}(D) = \mathcal{C}(D')$. In most applications requiring minimal OBDDs, it is enough to consider OBDDs that are minimal up to renaming of states. In our applications however, syntactic uniqueness is an important requirement.

3.2 Second Order Finite Automata

Recall that we view an OBDD D of width w and length n as a sequence $D = B_1B_2\ldots B_n$ of w-layers. In other words, D is simply a string of length n over the alphabet $\widehat{\mathcal{B}}(w)$ of all layers. We note that not every string in the set $\widehat{\mathcal{B}}(w)^n$ is a valid OBDD of length n and width w, since not all such strings satisfy Conditions 1-3 stated in Subsection 3.1. We let $\widehat{\mathcal{B}}(w)^{\circ n}$ denote the set of all strings in $\widehat{\mathcal{B}}(w)^n$ that *are* OBDDs. We let $\widehat{\mathcal{B}}(w)^{\circledast} = \bigcup_{n\in\mathbb{N}} \widehat{\mathcal{B}}(w)^{\circ n}$ denote the set containing the empty string, and all OBDDs of width at most w. The following proposition states that the set $\widehat{\mathcal{B}}(w)^{\circledast}$ is 2^w-regular.

Proposition 4. *For each $w \in \mathbb{N}$, there is a finite automaton $\mathcal{A}(w)$ with 2^w states such that $\mathcal{L}(\mathcal{A}(w)) = \widehat{\mathcal{B}}(w)^{\circledast}$.*

Since each OBDD D of width at most w represents a function f_D, a set of OBDDs of width at most w may be seen as a representation of a function class. That is to say, the class consisting of all functions associated with OBDDs in the set. We will be particularly concerned with function classes that can be finitely represented using finite automata over the alphabet $\widehat{\mathcal{B}}(w)$.

Definition 5. *We say that a finite automaton $\mathcal{A}$ over $\widehat{\mathcal{B}}(w)$ is a* second-order finite automaton *if $\mathcal{L}(\mathcal{A}) \subseteq \widehat{\mathcal{B}}(w)^{\circledast}$. We let*

$$\mathcal{F}(\mathcal{A}) = \{f_D \ : \ D \in \mathcal{L}(\mathcal{A})\}$$

be the function class represented by $\mathcal{A}$.

We say that a function class $\mathcal{G}$ is *decisional* if there is some $w \in \mathbb{N}$ and some second-order finite automaton $\mathcal{A}$ over $\widehat{\mathcal{B}}(w)$ such that $\mathcal{G} = \mathcal{F}(\mathcal{A})$.

Let $w,w' \in \mathbb{N}$, and let $\mathcal{G} \subseteq \mathcal{F}(\mathcal{A}(w))$. We define the width-$w$ complement of $\mathcal{G}$ as the function class $\overline{\mathcal{G}}^w = \mathcal{F}(\mathcal{A}(w))\setminus\mathcal{G}$. The next theorem, from (de Melo and de Oliveira Oliveira 2022), states some basic closure and decidability properties for decisional function classes. We note that these properties do not follow directly from usual closure and decidability properties of regular languages. The problem is that several OBDDs of width at most w may represent the same function. Therefore, it is not in general possible to establish a one-to-one correspondence between functions in $\mathcal{F}(\mathcal{A})$ and OBDDs in $\mathcal{L}(\mathcal{A})$.

Theorem 6. *Regular function classes are closed under union, intersection, and bounded-width complementation. Furthermore, inclusion and intersection emptiness for regular function classes are decidable.*

4 Functional Reconfiguration

A Boolean circuit C with k inputs may be regarded as the specification of a process that *reconfigures* input functions $f_1, \ldots, f_k$ into the output function $f = F_C(f_1, \ldots, f_k)$, where F_C is the k-bit function computed by C. In this section, we introduce a suitable measure of complexity for such reconfiguration process. More specifically, we let the *reconfiguration width* of C with respect to $f_1, \ldots, f_k$ be the maximum complexity (i.e. width) of a function $F_{g_i}(f_1, \ldots, f_k)$ arising during the reconfiguration process.

Given a function $f : \{0,1\}^n \to \{0,1\}$ we let $\mathsf{Can}(f)$ denote the (unique) canonical OBDD D of length n such that $f_D = f$ (see Theorem 3). In other words, the canonical OBDD D with language $\mathcal{L}(D) = f^{-1}(1)$. We note that D has minimum width among all OBDDs representing f. Using this notion, we can define the notion of *reconfiguration width* of a k-input Boolean circuit C with respect to functions $f_1, \ldots, f_k : \{0,1\}^n \to \{0,1\}$.

Definition 7 (Reconfiguration Width)**.** *Let $C = (g_1, \ldots, g_m)$ be a Boolean circuit with k inputs, and $f_1, \ldots, f_k : \{0,1\}^n \to \{0,1\}$ be Boolean functions. The reconfiguration width of C with respect to $f_1, \ldots, f_k$ is defined as*

$$\mathbf{w}(C, f_1, \ldots, f_k) = \max_{i \in [m]} \mathrm{w}(\mathsf{Can}(F_{g_i}(f_1, \ldots, f_k))).$$

The next problem is a refinement of FUNCTIONALDECOMPOSITION . In this refinement, we require that the reconfiguration width of C with respect to $f_{D_1}, \ldots, f_{D_k}$ is bounded by w. In other words, one wishes to decompose a function f_D into simpler functions $f_{D_1}, \ldots, f_{D_k}$ in such a way that all functions occurring in the process of reconfiguring $f_{D_1}, \ldots, f_{D_k}$ into f_D have low complexity (i.e. width at most w).

Problem name: FUNCTIONALRECONFIGURATION
Given: Numbers $k, w, p, m \in \mathbb{N}$ with $p < w$, an OBDD D, a Boolean circuit C with k inputs and m gates, and a second-order finite automaton $\mathcal{A}$ over $\widehat{\mathcal{B}}(p)$.
Parameters: k, w, m,$|\mathcal{A}|$.
Question: Are there OBDDs $D_1, \ldots, D_k \in \mathcal{F}(\mathcal{A})$ of width at most p such that $f_D = F_C(f_{D_1}, \ldots, f_{D_k})$ and $\mathbf{w}(C, f_{D_1}, \ldots, f_{D_k}) \leq w$?

The next theorem, whose proof can be found in the full version of this work, states that FUNCTIONALRECONFIGURATION is fixed-parameter tractable when parameterized by the number of factor functions k, the reconfiguration width w, the size m of the circuit C, and the size of the second-order automaton representing the class of functions $|\mathcal{A}|$. We note that the parameter m is upper bounded by $O(2^k/k)$ since any Boolean function on k inputs can be computed by a circuit of size at most $O(2^k/k)$. Additionally, the dependency on the parameter $|\mathcal{A}|$ is of the form $|\mathcal{A}|^k$. Therefore when k and w are fixed, our algorithm for FUNCTIONALRECONFIGURATION runs in polynomial time even if the number of states of $\mathcal{A}$ is polynomial in n.

Theorem 8. FUNCTIONALRECONFIGURATION *can be solved in time*

$$2^{\mathcal{O}(m \cdot w^2 \log w)} \cdot |\mathcal{A}|^k \cdot |D|.$$

In the problem FUNCTIONALRECONFIGURATION , we need to specify the maximum width of an OBDD occurring during the reconfiguration process. Next, we estimate an upper bound for reconfiguration width in terms of the maximum width p of the canonical OBDD representing one of the factor functions and the depth d of the decomposer circuit C. By noting that d is upper bounded by the number of gates m of the circuit, and that m is upper bounded by $O(2^k/k)$ we also obtain bounds for solving FUNCTIONALDECOMPOSITION parameterized by p and m or by p and k.

Lemma 9. *Let $D_1, \ldots, D_k$ be OBDDs in $\widehat{\mathcal{B}}(p)^{\circ n}$, and let C be a circuit with k inputs and depth at most d. Then $\mathbf{w}(C, f_{D_1}, \ldots, f_{D_k}) \leq p^{2^d}$.*

Proof. Let $C = (g_1, g_2, \ldots, g_m)$. We claim that for each $i \in [m]$, the width of the OBDD $\mathsf{Can}(F_{g_i}(f_{D_1}, \ldots, f_{D_k}))$ is upper-bounded by p^{2^d}, where d is the depth of gate g_i in C. The proof is by induction on the depth of a gate. In the base case, let g_i be a gate of depth 0, that is to say, an input gate. Then $F_{g_i}(f_{D_1}, \ldots, f_{D_k}) = f_{D_j}$ for some $j \in [k]$. Since, $D_j \in \widehat{\mathcal{B}}(p)^{\circ n}$, we have that the width of $\mathsf{Can}(F_{g_i}(f_{D_1}, \ldots, f_{Dk}))$ is at most $p^{2^0} = p$. Now, let $g_i = \mathtt{AND}(g_j, g_k)$ be a gate of depth d, and suppose that the statement of the lemma holds for every gate of depth at most $d-1$. Then by the induction hypothesis, there are OBDDs $D_{g_j}, D_{g_k} \in \widehat{\mathcal{B}}(p^{2^{d-1}})$ computing the functions $F_{g_j}(f_{D_1}, \ldots, f_{Dk})$ and $F_{g_k}(f_{D_1}, \ldots, f_{D_k})$ respectively. Since $g_j = \mathtt{AND}(g_j, g_k)$, we have that $f_{D_{g_i}} = f_{D_{g_j}} \wedge f_{D_{g_k}}$, or alternatively, $\mathcal{L}(D_{g_i}) = \mathcal{L}(D_{g_j}) \cap \mathcal{L}(D_{g_k})$. It can be shown (see Lemma 15 of (Andrade de Melo and de Oliveira Oliveira 2019)) that given OBDDs D and D' of width at most p there is an OBDD of width at most p^2 accepting the language $\mathcal{L}(D) \cap \mathcal{L}(D')$. Therefore, we have that there is an OBDD of width at most $(p^{2^{d-1}})^2 = p^{2^d}$ computing the function $F_{g_i}(f_{D_1}, \ldots, f_{D_k})$. A similar argument can be proved in the case that $g_i = \mathtt{OR}(g_i, g_j)$. □

By plugging Lemma 9 into Theorem 8 we get the following upper bounds for solving FUNCTIONALDECOMPOSITION .

Theorem 10. FUNCTIONALDECOMPOSITION *can be solved in time*

1. $2^{p^{O(2^d)}} \cdot |\mathcal{A}|^k \cdot |D|$,
2. $2^{p^{O(2^m)}} \cdot |\mathcal{A}|^k \cdot |D|$,
3. $2^{p^{2^{O(2^k/k)}}} \cdot |\mathcal{A}|^k \cdot |D|$.

Proof. Item 1 follows directly by plugging Lemma 9 into Theorem 8. Item 2 follows by noting that a circuit of size m has depth at most m. Item 3 follows from the fact that any Boolean function with k inputs can be computed by a circuit of size $O(2^k/k)$. □

5 Applications

In this section we describe some prominent application for the FUNCTIONALDECOMPOSITION problem. The first applications is concerned with the notion of a generalized k-junta. That is a function that is a Boolean combination of k functions belonging to some class $\mathcal{F}$. The crucial difference is that here, the decomposer circuit C is not given a priori. In the second application we obtain an FPT algorithm for the OBDD factorization problem, parameterized by the number of factor OBDDs and the width of the OBDDs. This is problem is analogous to the DFA factorization problem (Jecker, Mazzocchi, and Wolf 2021; Jecker, Kupferman, and Mazzocchi 2020; Kupferman and Mosheiff 2013), which does not admit and FPT algorithm parameterized by the number of factor automata.

5.1 Generalized Juntas

For $n, i \in \mathbb{N}$, with $i \leq n$, the (n, i)-hypercube is the function $H_{n,i} : \{0,1\}^n \to \{0,1\}$ such that for each $x \in \{0,1\}^n$, $H_{n,i}(x) = 1$ if and only if $x_i = 1$. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be a k-junta if there are numbers $j_1, \ldots, j_k$ such that f is a Boolean combination of $H_{n,j_1}, \ldots, H_{n,j_k}$. Intuitively, f is a k-junta if there are coordinates $j_1, \ldots, j_k \in [n]$ such that for each input $x \in \{0,1\}^n$, the value $f(x)$ is determined by the values of $x_{j_1}, \ldots, x_{j_k}$. The problem of determining whether a function is a k-junta has been studied under a wide variety of contexts (Blais 2010). It turns out that when the input function f is specified by an OBDD D, then for each $k \in \mathbb{N}$, there is a simple algorithm that decides whether f is a k junta in time linear on the size of D.

Observation 11. *Given an OBDD D of length n and a number $k \in \mathbb{N}$, one can determine whether the function $f_D : \{0,1\}^n \to \{0,1\}$ is a k-junta in time linear on the size of D.*

Proof. We assume that D is minimized. Otherwise, we just apply the standard minimization algorithm for OBDDs. Let $i \in [n]$ and suppose that f_D does not depend on the value of x_i. Then, every transition from any state q of layer i to layer $i+1$ must be of the form $(q, 0, q')$ and $(q, 1, q')$. In other words, setting x_i to 1 or 0 from state q should both lead to state q'. If this were not the case, that is, if there were transition $(q, 0, q')$ and $(q, 1, q'')$ in layer i with $q' \neq q''$, then q' and q'' could be merged since they accept the same words (because f_D does not depend on x_i). This would be in contradiction with the assumption that D is minimized. Therefore, given a minimized OBDD, one can just traverse each layer checking for each layer i whether it contains transitions $(q, 0, q')$ and $(q, 1, q'')$ for $q' \neq q''$. Doing so, we identify the set of variables $Y \subseteq x_1, \ldots, x_n$ on which D depends. Deciding whether D is a k-Junta hence boils down to check whether $|Y| \leq k$. □

More general notions of juntas have been considered in the literature under a variety of contexts (De, Mossel, and Neeman 2021, 2019). In these contexts, instead of requiring the k factor functions $f_1, \ldots, f_k$ to be hypercubes, we require that these functions belong to some well-behaved class of functions $\mathcal{F}$.

Definition 12. *Let $\mathcal{F}$ be a class of Boolean functions and $k \in \mathbb{N}$. We say that a function $f : \{0,1\}^n \to \{0,1\}$ is a $(k, \mathcal{F})$-junta if there are k functions $f_1, \ldots, f_k : \{0,1\}^n \to \{0,1\}$ in $\mathcal{F}$ such that f is a Boolean combination of $f_1, \ldots, f_k$.*

Note that equivalently, $f : \{0,1\}^n \to \{0,1\}$ is a $(k, \mathcal{F})$-junta, if it admits a $(C, \mathcal{F})$-decomposition $f_1, \ldots, f_k : \{0,1\}^n \to \{0,1\}$ for some circuit C. This motivates the definition of a variant of FUNCTIONALDECOMPOSITION where instead of providing specifying a composition circuit C we simply specify an upper bound on the size of such a circuit. We call this generalization GENERALIZEDJUNTA .

Problem name: GENERALIZEDJUNTA
Given: Numbers $k, p, m \in \mathbb{N}$, an OBDD D, and a second-order finite automaton $\mathcal{A}$ over $\widehat{\mathcal{B}}(p)$.
Parameters: k, p, m, $|\mathcal{A}|$.
Question: Is there a Boolean circuit C with k inputs and at most m gates, and OBDDs $D_1, \ldots, D_k \in \mathcal{A}$ of width at most p such that $f_D = F_C(f_{D_1}, \ldots, f_{D_k})$?

The next theorem states that GENERALIZEDJUNTA can be solved within the same asymptotic bounds provided in Theorem 10.

Theorem 13. GENERALIZEDJUNTA *can be solved in time*

1. $2^{p^{O(2^d)}} \cdot |\mathcal{A}|^k \cdot |D|$,
2. $2^{p^{O(2^m)}} \cdot |\mathcal{A}|^k \cdot |D|$,
3. $2^{p^{2^{O(2^k/k)}}} \cdot |\mathcal{A}|^k \cdot |D|$.

Proof. The theorem follows by enumerating all possible circuits of size at most m and then applying Theorem 10. Given that a circuit with m gates can be specified with $O(m \log m)$ bits, we have that the total number of such circuits is $2^{O(m \log m)}$. Given that $2^{O(m \log m)} \cdot 2^{p^{O(2^m)}} \cdot |\mathcal{A}|^k \cdot |D|$ is still $2^{p^{O(2^m)}} \cdot |\mathcal{A}|^k \cdot |D|$, we have proved Item 2. If we want to parameterize the problem only by k and p, then we just need to enumerate all circuits with up to $m = O(2^k/k)$ gates. Therefore, we have Item 3. On the other hand, if we want to parameterize only by the depth d of the circuit then we note that any circuit of depth d has at most $2^{O(d)}$ gates. Therefore, the complexity parameterized by depth is $2^{2^{O(d)}} \cdot 2^{p^{O(2^d)}} \cdot |\mathcal{A}|^k \cdot |D|$ which is still asymptotically $2^{p^{O(2^d)}} \cdot |\mathcal{A}|^k \cdot |D|$. Therefore, we have Item 1. □

5.2 OBDD Factorization

A prominent factorization problem in the context of automata theory is the *DFA factorization problem* (Jecker, Mazzocchi, and Wolf 2021; Jecker, Kupferman, and Mazzocchi 2020; Kupferman and Mosheiff 2013). In this problem we are given a DFA A with w states and the goal is to

determine whether there are DFAs $A_1, \ldots, A_k$ each with at most $w-1$ states such that $L(A) = L(A_1) \cap \cdots \cap L(A_k)$.

An analogous problem in the context of OBDDs, the OBDD factorization problem, we are given an OBDD of width at most w and the goal is to determine whether there exist OBDDs $D_1, \ldots, D_k$ each of width at most $w-1$ such that $L(D) = L(D_1) \cap \cdots \cap L(D_k)$. A direct consequence of our main theorem is that this problem can be solved in fixed parameter linear time (that is, linear in $|D|$) by instantiating the second-order automaton $\mathcal{A}$ as the automaton that accepts all OBDDs of width at most $w-1$.

Corollary 14. *Given an OBDD D of length n and width at most w, one can determine in time $2^{w^{O(2^k)}} \cdot |D|$, whether there are OBDDS $D_1, \ldots, D_k$ of width at most $w-1$ such that $L(D) = L(D_1) \cap \cdots \cap L(D_k)$.*

6 Conclusion and Related Work

In this work we introduced a new framework to address functional reconfiguration and functional decomposition problems by combining techniques from automata theory and parameterized complexity theory. In our results, we assume that the function $f : \{0,1\}^n \to \{0,1\}$ is specified explicitly at the input using an OBDD D. We note that this assumption can be relaxed if instead of an OBDD we are given access to an oracle that is able to answer membership and equivalence queries with respect to the language $L = \{x \in \{0,1\}^n \ : \ f(x) = 1\}$. More specifically, in the traditional model of learning with membership and equivalence queries, the objective is to learn a regular language L over a given alphabet Σ, where the learner interacts with an oracle (a minimally adequate teacher) capable of answering two types of queries:

1. *Membership queries*: the learner selects a word in $x \in \Sigma^*$ and the teacher replies whether or not $x \in L$.
2. *Equivalence queries*: the learner selects an hypothesis automaton H and the teacher replies whether or not L is the language of H. If this is not the case, the teacher provides the learner with a counter-example word $x \in \Sigma^* \backslash L$.

A classic result due to Angluin (Angluin 1987) states that any regular language L can be exactly learned with a number of queries that is polynomial in the number of states of a minimum deterministic automaton representing L, and in the size of the largest counter-example returned by the teacher. Since an OBDD of length n and width w may be regarded as an acyclic finite automaton with $O(w \cdot n)$ states, and since one can assume in this case that counter-examples are of size $O(n)$, Angluin's theorem implies the following corollary.

Corollary 15. *Let $L \subseteq \{0,1\}^n$ be a language with the property that there is an OBDD $D \in \widehat{\mathcal{B}}(w)^{\circledast}$ with $L = L(D)$. Then one can learn L with $w^{O(1)} \cdot n^{O(1)}$ membership and equivalence queries.*

Therefore, in view of Corollary 15 our results (Theorem 8, Theorem 10 and Theorem 13) can also be used to provide a parameterized approach for approaching the problems FUNCTIONALDECOMPOSITION , FUNCTIONALRECONFIGURATION and GENERALIZEDJUNTA even if an explicit description of the target OBDD to be decomposed is not known, as long as we do have an oracle that is able to answer membership and equivalence queries.

Acknowledgments

We acknowledge support from the Research Council of Norway, grant numbers 288761 and 326537.

References

Andrade de Melo, A.; and de Oliveira Oliveira, M. 2019. On the Width of Regular Classes of Finite Structures. In Fontaine, P., ed., *Automated Deduction – CADE 27*, 18–34. Cham: Springer International Publishing. ISBN 978-3-030-29436-6.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2): 87–106.

Atzmon, Y.; Kreuk, F.; Shalit, U.; and Chechik, G. 2020. A causal view of compositional zero-shot recognition. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Blais, E. 2010. Testing juntas: A brief survey. *Property Testing: Current research and surveys*, 32–40.

Bohanec, M.; and Zupan, B. 2004. A function-decomposition method for development of hierarchical multi-attribute decision models. *Decision Support Systems*, 36(3): 215–233.

Bollig, B. 2012. On symbolic OBDD-based algorithms for the minimum spanning tree problem. *Theoretical Computer Science*, 447: 2–12.

Bollig, B. 2014. On the width of ordered binary decision diagrams. In *International Conference on Combinatorial Optimization and Applications*, 444–458. Springer.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3): 293–318.

Case, J.; Jain, S.; and Stephan, F. 2013. Automatic functions, linear time and learning. *Logical Methods in Computer Science*.

Correa, C. G.; Ho, M. K.; Callaway, F.; Daw, N. D.; and Griffiths, T. L. 2023. Humans decompose tasks by trading off utility and computational cost. *PLoS computational biology*, 19(6): e1011087.

De, A.; Mossel, E.; and Neeman, J. 2019. Is your function low dimensional? In *Conference on Learning Theory*, 979–993. PMLR.

De, A.; Mossel, E.; and Neeman, J. 2021. Robust testing of low dimensional functions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 584–597.

de Melo, A. A.; and de Oliveira Oliveira, M. 2022. Second-Order Finite Automata. *Theory Comput. Syst.*, 66(4): 861–909.

Deniziak, S.; and Wiśniewski, M. 2021. DECOLib: A library of components for DECOmposition of discrete functions. *SoftwareX*, 16: 100799.

Goel, A. K.; and Chandrasekaran, B. 1989. Functional Representation of Designs and Redesign Problem Solving. In Sridharan, N. S., ed., *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, 1388–1394. Morgan Kaufmann.

Hachtel, G. D.; and Somenzi, F. 1993. A symbolic algorithm for maximum flow in 0-1 networks. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, 403–406. IEEE Computer Society Press.

Hashemi, A.; and Vikalo, H. 2018. Evolutionary self-expressive models for subspace clustering. *IEEE Journal of Selected Topics in Signal Processing*, 12(6): 1534–1546.

Hiabu, M.; Meyer, J. T.; and Wright, M. N. 2023. Unifying local and global model explanations by functional decomposition of low dimensional structures. In *International Conference on Artificial Intelligence and Statistics*, 7040–7060. PMLR.

Hopcroft, J. 1971. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, 189–196. Elsevier.

Izza, Y.; Jabbour, S.; Raddaoui, B.; and Boudane, A. 2020. On the Enumeration of Association Rules: A Decomposition-based Approach. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 1265–1271. ijcai.org.

Jain, S.; Luo, Q.; and Stephan, F. 2012. Learnability of automatic classes. *Journal of Computer and System Sciences*, 78(6): 1910–1927.

Jecker, I.; Mazzocchi, N.; and Wolf, P. 2021. Decomposing Permutation Automata. In Haddad, S.; and Varacca, D., eds., *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Jecker, I. R.; Kupferman, O.; and Mazzocchi, N. 2020. Unary prime languages. In *45th International Symposium on Mathematical Foundations of Computer Science*, volume 170.

Kupferman, O.; and Mosheiff, J. 2013. Prime Languages. In Chatterjee, K.; and Sgall, J., eds., *Mathematical Foundations of Computer Science 2013*, 607–618. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-40313-2.

Kuske, D. 2021. Second-Order Finite Automata: Expressive Power and Simple Proofs Using Automatic Structures. In Moreira, N.; and Reis, R., eds., *Proc. of the 25th International Conference on Developments in Language Theory (DLT 2021)*, volume 12811 of *Lecture Notes in Computer Science*, 242–254. Springer.

Laberge, G.; Pequignot, Y. B.; Marchand, M.; and Khomh, F. 2024. Tackling the XAI Disagreement Problem with Regional Explanations. In *International Conference on Artificial Intelligence and Statistics*, 2017–2025. PMLR.

Molnar, C.; Casalicchio, G.; and Bischl, B. 2020. Quantifying model complexity via functional decomposition for better post-hoc interpretability. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, 193–204. Springer.

Sawitzki, D. 2004. Implicit flow maximization by iterative squaring. In *International Conference on Current Trends in Theory and Practice of Computer Science*, 301–313. Springer.

Srivastava, S. 2023. Hierarchical Decompositions and Termination Analysis for Generalized Planning. *J. Artif. Intell. Res.*, 77: 1203–1236.

Sun, L.; Nguyen, C. H.; and Mamitsuka, H. 2019. Multiplicative Sparse Feature Decomposition for Efficient Multi-View Multi-Task Learning. In Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 3506–3512. ijcai.org.

Vykhovanets, V. S. 2006. Algebraic decomposition of discrete functions. *Automation and Remote Control*, 67(3): 361–392.

Wang, H.; Yang, M.; Wei, K.; and Deng, C. 2023. Hierarchical Prompt Learning for Compositional Zero-Shot Recognition. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, 1470–1478. ijcai.org.

Woelfel, P. 2006. Symbolic topological sorting with OBDDs. *Journal of Discrete Algorithms*, 4(1): 51–71.

Zaid, F. A.; Grädel, E.; and Reinhardt, F. 2017. Advice Automatic Structures and Uniformly Automatic Classes. In Goranko, V.; and Dam, M., eds., *Proc. of the 26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *LIPIcs*, 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Zupan, B.; Bohanec, M.; Bratko, I.; and Demsar, J. 1997. Machine learning by function decomposition. In *ICML*, 421–429.

Zupan, B.; Bratko, I.; Bohanec, M.; and Demšar, J. 1999. Function decomposition in machine learning. In *Advanced Course on Artificial Intelligence*, 71–101. Springer.