The Future is Hybrid: Next Generation Data Structures for Formal Verification

(Invited Paper)

Rolf Drechsler*†
* Group of Computer Architecture
University Bremen
Bremen, Germany
drechsler@uni-bremen.de

Christina Plump†
† Cyber-Physical Systems
DFKI Bremen
Bremen, Germany
christina.plump@dfki.de

Martha Schnieber*
* Group of Computer Architecture
University Bremen
Bremen, Germany
schnieber@uni-bremen.de

***Abstract*—Trust in electronic devices is dependent on their safe and reliable behavior. An integral part is the correct design of the hardware. While classically simulation-based approaches have been applied, only through formal proof techniques complete correctness can be guaranteed. The core of these formal approaches, and responsible for time and space complexity, is the choice of the underlying data structure to represent the functional behavior. A significant class of data structures are graph-based function representations, like BDDs, KFDDs or *BMDs. These have shown excellent properties – provability in polynomial time and space – for some function classes , e.g., adders. Experimental studies have validated these properties, and formal proofs can guarantee this behavior. Unfortunately, these properties often cannot be generalized to varying function classes. One reason is that graph-based representations are usually tailored for either bit-level or word-level functions. However, designing hybrid data structures that can represent both types in parallel might allow formal proofs for even larger functional classes.**

In this paper, we demonstrate how to design these hybrid data structures, overcoming limitations of current formal verification approaches. We introduce a generalized concept on decompositions and graph-based function representations based on Kronecker matrices with an extended element space and dimension. It is shown how these extensions allow the representation of hybrid function classes, paving the way for more trust in electronic devices.

***Index Terms*—Binary Decision Diagrams (BDD), Formal Verification, Hybrid Representation Forms**

I. Introduction

In the past decades, electronic devices and digital circuits have become ubiquitous in every day life. The complexity and size of digital circuits have grown significantly, increasing the need to systematically ensure correctness. If digital systems are not verified and errors are not detected prior to distribution, the consequences can be tremendous as multiple examples have shown over the years, e.g., the Intel Pentium bug in 1994, costing 475 million [1], Intel's Sandy Bridge chipset in 2011, costing 1 billion [2], or a security flaw in Apple's M-Series CPUs [3], which was recently discovered this year.

At the heart of every electronic device lies the *Processing Unit*, controlling its behaviour and performing all computations. As more and more *Processing Units* with a rising complexity are being developed nowadays, their verification is crucial to avoid costly errors. Thus, verification techniques have been developed in research and industry. Classically, there are several different techniques for achieving verified chips: they can be simulated, emulated, or formally verified. A mere simulation or emulation of chips results in low verification coverage, as only a few cases can be simulated and compared to the specified behavior for the PU. Thus, simulation and emulation can never completely ensure a PU's correctness. However, the *Formal Verification* (FV) of a chip proves its 100% correctness. One of the central tasks is equivalence checking, which is widely used and well-accepted in the industry. Thus, several commercial tools exist based on multi-engine solvers [4], but resources in terms of runtime and memory requirements are unknown prior to the actual verification process. This is not surprising since the underlying theoretical problem has been well analyzed and proven to be coNP-complete and, as such, generally requires exponential time and space. Even worse, equivalence checking for sequential circuits is EXPSPACE-complete [5] and, therefore, even more complex. Thus, while developing arbitrary PUs, it is impossible to predict whether or not a formal verification will be possible within given resources due to the unpredictability of resources. This trade-off between time and coverage has been a longstanding challenge for ensuring systems' correctness.

Nonetheless, using the right data structure to represent the circuit allows performing formal verification in reasonable (i.e., polynomial) time. In formal verification, data structure refers to the formal representation of a function. To formally verify a circuit, data structures can be leveraged by comparing the data structure instance representing the specified function with the data structure instance synthesized from the circuit itself (see Figure 1).

A significant class of data structures are graph-based function representations, like *Binary Decision Diagrams* (BDDs) [6], *Kronecker Functional Decision Diagram* (KFDDs) [7], or *Multiplicative Binary Moment Diagrams* (*BMDs) [8]. It has been shown that they have excellent properties for certain function classes, e.g., BDDs can verify adders in polynomial resources [9]–[11]. However,

This work was supported by the German Research Foundation (DFG) within the Reinhart Koselleck Project PolyVer (DR 287/36-1).

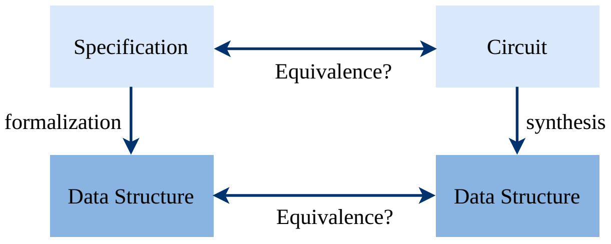

Fig. 1. Importance of Data Structures in Formal Verification

generalising these properties to varying function classes proves challenging. For example, while showing polynomial resources for adders, BDDs show exponential behaviour for multipliers [12]. *BMDs, however, allow polynomial upper bounds for specific types of multipliers [13]. These differences in verification effort stem from the fact that, for example, adders are bit-level functions, while multiplication can be represented more efficiently as a word-level function.

However, with the rise of new types of processing units, e.g., GPUs or AI-accelerating PUs like Google's TPU [14], complex modules are becoming more crucial for these PUs, like *Multiply-Accumulate functions* (MACs) which are essential for many graphics or AI-related computations. Computations on neural networks, for example, consist mainly of MACs. These now require a data structure for verification, that can deal with both: addition and multiplication, and hence, requires the combination of bit-level and word-level representations.

In this paper, we showcase how the design of hybrid data structures can overcome these barriers and allow data structures that are capable of representing both bit- and word-level functions simultaneously. We generalise hybrid data structures from the literature like *Kronecker Multiplicative Binary Moment Diagrams* (K*BMDs) [15] to show how extending a matrix representation format can lead to hybrid data structures that may allow polynomial verification in the future.

In the remainder of this paper, we will first remind the reader of different classes of graph-based data structures in the background section. In Section III, we refer to Kronecker products of non-singular matrices as a way of representing these graph-based structures. We then use this concept to generalise these graph-based data structures to work on hybrid functions in Section IV and conclude our approach in Section V.

II. Background

In this section, we shortly introduce classic bit-level and word-level diagrams.

A. Bit-level diagrams

At the beginning of all considerations regarding graph-based representations of functions, are BDDs, the basic data structure introduced by Bryant [6] in the mid-80s.

Definition 1. *BDDs are directed, acyclic graphs with a root $G = (V, E)$, with a set of nodes consisting of non-terminal and terminal nodes $V = T \cup N$. Non-terminal nodes v are labeled with a variable $label(v) \in \{x_1, x_2, \dots, x_n\}$, while terminal nodes are labeled from the set of Boolean values $label(t) \in \mathbb{B}$.*

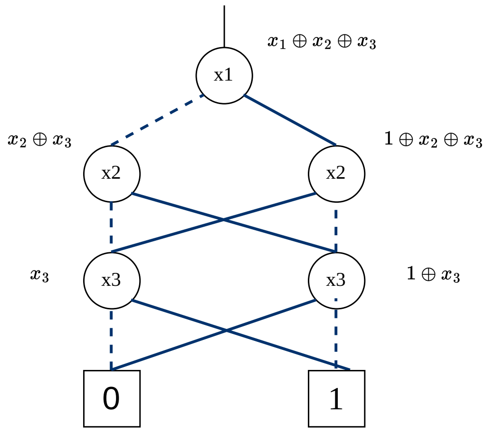

Fig. 2. A reduced BDD with natural order for the XOR-function

BDDs use the Shannon decomposition to represent Boolean functions $f : \mathbb{B}^n \to \mathbb{B}$, which works as follows:

$$\begin{aligned} f(v) &= \overline{x}_i \cdot f_{low(v)} + x_i \cdot f_{high(v)} \\ &= \overline{x}_i \cdot f_{x_i=0}(v) + x_i \cdot f_{x_i=1}(v) \end{aligned}$$

where $label(v) = x_i$ and $f_{x_i=b}(v)$ represents the function at the node v where x_i is set to $b \in \mathbb{B}$, i.e., the positive or negative cofactor of the function.

BDDs are called ordered, if the variables $\{x_1, \dots, x_n\}$ are only used as labels in the same order, i.e., there exists a mapping $\pi : \{1, \dots, n\} \to \{x_1, \dots, x_n\}$ such that $\pi^{-1}(label(low(v))) > \pi^{-1}(label(v))$ (equivalently for $high(v)$) for all non-terminal nodes v with non-terminal children $low(v), high(v)$.

Two nodes which have identical children, i.e., nodes v_1, v_2 with $low(v_1) = low(v_2)$ and $high(v_1) = high(v_2)$ are called *isomorphic* and can be merged. A node whose children are identical, i.e., a node v with $low(v) = high(v)$ corresponds to an irrelevant variable (setting $label(v)$ to either 0 or 1 results in the same function), is thus called *redundant* and can be removed [6].

A BDD that is ordered and has no isomorphic or redundant nodes (called reduced) is a canonical representation of its described Boolean function (given the respective order), which is one of the reasons why BDDs are widely used as a data representation structure for equivalence checking.

Figure 2 shows a reduced BDD for the XOR-function $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ and the inner functions represented by the nodes in the BDD.

However, the Shannon decomposition is not the only single-variable decomposition for Boolean functions. It has been shown in [16] that besides the Shannon decomposition only two more relevant single-variable decomposition types exist for Boolean functions when considering complemented edges: the positive and negative Davio decomposition. In [7], a

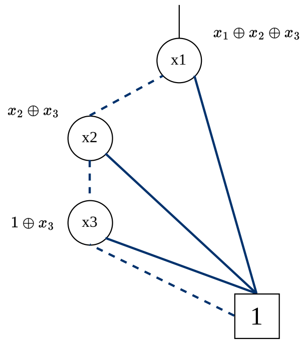

Fig. 3. A reduced KFDD with natural order for the XOR-function

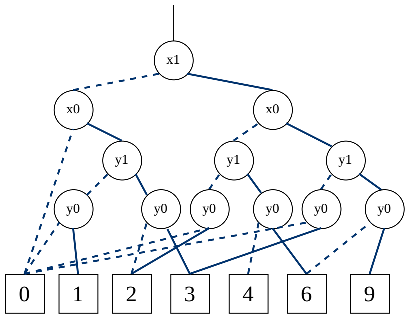

Fig. 4. A two-bit integer multiplication represented by an MTBDD

decision diagram type has been introduced that allows all three decomposition types, KFDDs.

Definition 2. *KFDDs are directed, acyclic graphs with a root $G = (V, E)$, with a set of nodes consisting of non-terminal and terminal nodes $V = T \cup N$. Non-terminal nodes v are labeled with a variable $label(v) \in \{x_1, x_2, \ldots, x_n\}$, while terminal nodes are labeled from the set of Boolean values $label(t) \in \mathbb{B}$. Additionally, they have a* Decomposition Type List *(DTL), which maps variables to either the Shannon (S), positive Davio (pD), or negative Davio (nD) decomposition.*

The Davio decompositions work as follows (with identical notation to the Shannon decomposition given above):

$$\begin{aligned} f^{pD}(v) &= f_{low(v)} \oplus x_i \cdot f_{high(v)} \\ &= f_{x_i=0}(v) \oplus x_i \cdot (f_{x_i=0}(v) \oplus f_{x_i=1}(v)) \qquad (1) \\ f^{nD}(v) &= f_{low(v)} \oplus \overline{x}_i \cdot f_{high(v)} \\ &= f_{x_i=1}(v) \oplus \overline{x}_i \cdot (f_{x_i=0}(v) \oplus f_{x_i=1}(v)) \end{aligned}$$

The notion of *ordered* is used equivalently to BDDs. Nodes with identical children are also called isomorphic and can be merged, and while the definition for redundance still refers to equal cofactors, its graphic representation for nodes that are decomposed with Davio differs from the Shannon decomposition: An inner function with equal cofactors will result in $f_{high(v)} = f_{x_i=0}(v) \oplus f_{x_i=1}(v) = 0$. Thus, for nodes that are decomposed with Davio, it holds that they can be removed when their high child is the 0-terminal.

Given an order and a decomposition type list, a reduced and ordered KFDD is a canonical representation of the corresponding Boolean function and therefore can be used as data representation structure for equivalence checking as well.

Figure 3 shows a KFDD for the XOR-function with decomposition type list (pD, nD, pD). One can easily see that leveraging these different decomposition types can lead to size reductions.

B. Word-level diagrams

Generally speaking, all arithmetic and logical functions can be represented by Boolean functions. Nevertheless, for some functions, it is more efficient to use a representation as a Pseudo-Boolean function (e.g., multiplication). For Pseudo-Boolean functions, i.e., $f : \mathbb{B}^n \rightarrow \mathbb{Z}$, one usually extends the terminal set in the above listed definitions.

The most straight-forward way to do so, are *Multi-Terminal BDDs* (MTBDDs) [17] or *Algebraic Decision Diagrams* (ADD) [18]. They simply extend the above definition for BDDs as follows:

Definition 3. *An MTBDD or ADD is a binary decision diagram whose terminal nodes can take on integer values, i.e., $G = (N \cup T, E)$ with $T \subset \mathbb{Z}$.*

All above mentioned notions apply here as well.

Another method relates to applying a variant of the positive Davio decomposition: The moment decomposition. It is defined as follows:

$$\begin{aligned} f^{M}(v) &= f_{low(v)} + x_i \cdot f_{high(v)} \\ &= f_{x_i=0}(v) + x_i \cdot (f_{x_i=1}(v) - f_{x_i=0}(v)) \qquad (2) \end{aligned}$$

Seeing $\oplus$ as Boolean difference, the relation of Equation (2) and Equation (1) becomes obvious. Diagrams using this decomposition are *Functional Decision Diagrams* (FDDs) [19] or *Binary Moment Diagrams* (BMDs) [8].

Another option to extend BDDs to be capable of representing word-level functions, is the addition of edge values as has been done with *Edge-Valued Binary Decision Diagrams* (EVBDDs) in [20]. Applying this idea to BMDs simplifies these representations by sharing common subexpressions, yielding the concept of Multiplicative Binary Moment Diagrams. A simple example showing the advantage of *BMDs over MTBDDs is multiplication. Figure 4 shows the multiplication of two two-bit integers represented by an MTBDD, while Figure 5 shows the multiplication represented by a *BMD leveraging shared subexpressions and the linear structure of the multiplication.

III. Representing data structures through Kronecker products of matrices

While the decomposition formulas presented in the last section, are a good way to represent a singular decomposition step, they are not as easily treatable, when having a look at several (or all) decomposition steps for a given function.

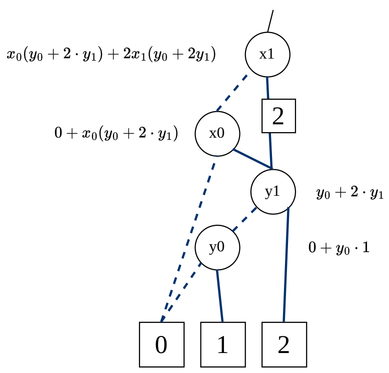

Fig. 5. A two-bit integer multiplication represented by a *BMD

In [21], results from [22] are leveraged to derive a matrix representation for function decomposition. It is shown that all three decomposition types can be represented as

$$\begin{pmatrix} f_{low(v)} \\ f_{high(v)} \end{pmatrix} = T \cdot \begin{pmatrix} f_{x_i=0} \\ f_{x_i=1} \end{pmatrix}$$

where $T \in GL_2(\mathbb{Z}) = \{A \in \mathbb{Z}^{2\times 2} | detA \neq 0\}$.

We obtain the following matrices:

$$T_S = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, T_{pD} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}, T_{nD} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}$$

when interpreting $\oplus$ as Boolean difference again.

This representation allows a recursive definition using the Kronecker product of matrices. As a quick reminder, the Kronecker product of two matrices A, B, is defined as follows:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \otimes B = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix}$$
$$= \begin{pmatrix} a_{11}b_{11} & a_{12}b_{11} & a_{11}b_{12} & a_{12}b_{12} \\ a_{21}b_{11} & a_{22}b_{11} & a_{21}b_{12} & a_{22}b_{12} \\ a_{11}b_{21} & a_{12}b_{21} & a_{11}b_{22} & a_{12}b_{22} \\ a_{21}b_{21} & a_{22}b_{21} & a_{21}b_{22} & a_{22}b_{22} \end{pmatrix}$$

Using this Kronecker product, they give a representation for the terminals following every path as follows:

$$\begin{pmatrix} f_{low,low,\dots,low} \\ f_{low,low,\dots,high} \\ \vdots \\ f_{high,high,\dots,low} \\ f_{high,high,\dots,high} \end{pmatrix} = (T \otimes \dots \otimes T) \cdot \begin{pmatrix} f_{x_1=0,\dots,x_n=0} \\ f_{x_1=0,\dots,x_n=1} \\ \vdots \\ f_{x_1=1,\dots,x_n=0} \\ f_{x_1=1,\dots,x_n=1} \end{pmatrix} \quad (3)$$

In this formulation, the vectors have dimension 2^n for a function $f : \mathbb{B}^n \to \mathbb{B}$, and the notation $low, low, \dots, high$ implies the choice of child throughout the decision tree. As the Kronecker product of non-singular matrices is non-singular as well, inverting Equation (3) is possible as well. The Shannon decomposition matrix is the identity matrix, which maps to the point-decomposition nature of the decomposition. It additionally allows transferring this concept to MTBDDs without any adaption.

To be able to use this concept on KFDDs, it is necessary to allow T in Equation (3) to vary between T_S, T_{pD}, T_{nD}. As all matrices are from $GL(\mathbb{Z})$, their Kronecker product is as well and therefore, the transformation can be used as shown above.

Example 1. *Let's have a look at the KFDD in Figure 3. The variable ordering is natural and the decomposition type list is given as (pD, nD, pD). We'll first go through every decomposition step on its own. First, we have cofactors of x_1 with $f_{x_1=0} = 0 \oplus x_2 \oplus x_3$, and $f_{x_1=1} = 1 \oplus x_2 \oplus x_3$, so we have:*

$$\begin{pmatrix} f_{low(v_{x_1})} \\ f_{high(v_{x_1})} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \oplus x_2 \oplus x_3 \\ 1 \oplus x_2 \oplus x_3 \end{pmatrix} = \begin{pmatrix} x_2 \oplus x_3 \\ 1 \end{pmatrix}$$

giving us the respective inner functions for the child-nodes. To apply the negative Davio decomposition to nodes labeled with x_2, we first have to compute cofactors again:

$$f_{low(v_{x_1}),x_2=0} = 0 \oplus 0 \oplus x_3, f_{low(v_{x_1}),x_2=1} = 0 \oplus 1 \oplus x_3$$

for the left node, and

$$f_{high(v_{x_1}),x_2=0} = 1, f_{high(v_{x_1}),x_2=1} = 1$$

for the right node (which has already been reduced in Figure 3). Then, we have for the left node:

$$\begin{pmatrix} f_{low(low(v_{x_1}))} \\ f_{high(low(v_{x_1}))} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_3 \\ 1 \oplus x_3 \end{pmatrix} = \begin{pmatrix} 1 \oplus x_3 \\ 1 \end{pmatrix}$$

And for the right node:

$$\begin{pmatrix} f_{low(high(v_{x_1}))} \\ f_{high(high(v_{x_1}))} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

We can now see that the high child of the right node has 0 as cofactor, so it will be reduced (in the end) and does not appear in the reduced KFDD.

Now, we proceed to the third level and thus the third variable x_3 which is again decomposed with using the positive Davio decomposition. For the left-most path, we obtain the following cofactors: $f_{low(low(v_{x_1})),x_3=0} = 1 \oplus 0 = 1$ and $f_{low(low(v_{x_1})),x_3=1} = 1 \oplus 1 = 0$. This leads to:

$$\begin{pmatrix} f_{low(low(low((v_{x_1}))))} \\ f_{high(low(low((v_{x_1}))))} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

This last equation shows that both children of the node labeled with x_3 end at the 1-terminal.

Using the adaption of Equation (3), however, we could compute $T_{pD} \otimes T_{nD} \otimes T_{pD}$, and multiply it with the vector-representation of the XOR-function, i.e., $(0,1,1,0,1,0,0,1)^T$. This results in $(1,1,1,0,1,0,0,0)^T$, which is exactly the sequence of terminals in the respective KFDD before applying reductions.

IV. Going hybrid - extending Kronecker Matrices

We have seen in the last sections that the matrix representation is capable of capturing several graph-representation forms that focus on a single class of functions, i.e., they are either on the bit-level (BDDs, KFDDs), or on the word-level (MTBDDs).

For specific circuit types, different hybrid graph representations have been introduced, one of the most generalised ones being K*BMDs in [15]. K*BMDs leverage the advantages of *BMDs by allowing multiplicative edges, but extend them to (a) allow additive edge weights as well, and (b) allow all three decomposition types. They thus allow the following three decompositions:

$$
\begin{aligned}
<(a,m),f_v>_S &= a + m((1-x)f_{low(v)} + xf_{high(v)}) \\
<(a,m),f_v>_{pD} &= a + m((f_{low(v)} + xf_{high(v)})) \\
<(a,m),f_v>_{nD} &= a + m(f_{low(v)} + (1-x)f_{high(v)})
\end{aligned}
$$

Adding a set of restrictions to the edge weights, the canonical form can — once more — be ensured. Additionally, it leads to the two important properties for the reduced K*BMDs that make up for the additional required storage due to the weights: First, there is only one terminal needed, i.e., the 0-terminal. Second, every variable has only one associated node in the reduced K*BMD. Third, storing the low-edges is not necessary, as they are normed to $(0,1)$, i.e., an additive edge weight of $a = 0$ and an multiplicative edge weight of $m = 1$. There also is a related form of K*BMDs that modifies these edge restrictions and normalisation computations in such a way that all weights are integer weights, making the representation even more efficient.

The generalised concept of the decomposition formulas shows that all introduced DDs can be subsumed by this graph-representation. Figure 6 shows an example of the application of K*BMDs to the following function:

$$\texttt{if}(x_2 \oplus x_1 \oplus x_0)\ \texttt{then}\ Y \cdot Z\ \texttt{else}\ 0$$

This function combines a subfunction that is most efficiently represented on the bit-level ($x_2 \oplus x_1 \oplus x_0$) (see the light blue part) with a subfunction which is most efficiently represented on the world-level ($Y \cdot Z$) (see the light gray part). Please note, that — to improve readability — we have not completely reduced this decision diagram. K*BMDs have a decisive advantage over *BMDs: As they allow the Shannon-decomposition, they are also capable of representing functions that are solely on the bit-level. By restricting the multiplicative weights to $m = 1$, only allowing $a \in 0, 1$, and restricting to the Shannon decomposition, they are isomorphic to BDDs and are therefore as well suited for bit-level functions as BDDs. Word-level functions on the other hand can be represented at least as well as with *BMDs, by restricting to positive Davio decomposition. The greater expressive power, therefore, does not come with a loss of representation efficiency for simpler functions. In Figure 6, the bit-level representation is done for the `XOR` part of the function, which is a classical bit-level function. For the lower part, different values are used for the edges, to model the word-level multiplication properties.

This showcases that hybrid data structures can be used to efficiently represent functions on both levels.

To decompose a hybrid function into a K*BMD top-down, it is necessary to compute the inverse affine-linear transformation prior to the decomposition, i.e., given the ingoing function

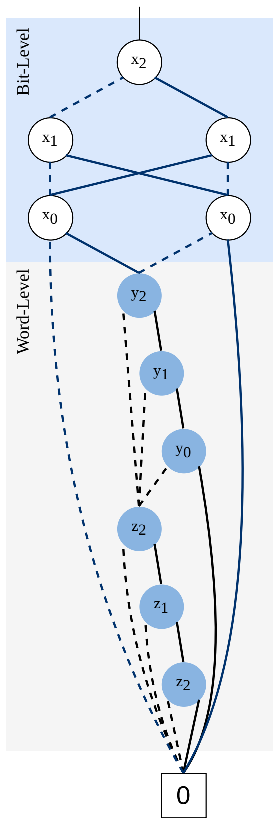

Fig. 6. K*BMD as example of a hybrid data structure

$f_v^{a,m}$ at a node v, $f(v) = (f^{a,m} - a)/m$ must be computed to apply the standard decompositions. Once the full K*BMD is computed, normalisation and reduction steps have to be performed to ensure canonicity. Please note, that it is possible that a reiteration of reduction and normalisation is necessary, i.e., a reduced K*BMD that undergoes a normalisation step might become unreduced again (and vice versa). We reach a canonical K*BMD exactly when there are no reduction steps or normalisation steps possible.

Using the Kronecker matrix representation introduced in the last section, K*BMDs can be decomposed with the same technique by allowing all three decomposition type matrices (T_S, T_{pD}, T_{nD}), assuming neutral weights (i.e., $a = 0$, $m = 1$) and allowing the function vector $f(b) \in \mathbb{Z}$. Then, it is possible to first compute the Kronecker product of the respective decomposition matrices, apply it to the function vector and obtain the terminals of the maximum K*BMD. Then, the respective reduction and normalisation steps can be applied to obtain a canonical representation.

We have therefore seen that the decomposition matrix technique which is applicable to many graph-based data structures, such as BDDs, KFDDs and MTBDDs, can be generalised and is thus also capable of capturing hybrid functions.

It is noteworthy that up until now, we have only used 2×2-matrices with entries from $\{1, -1, 0\}$. Starting from these matrices, there are several extension possibilities which might lead to new graph-based data representation structures.

One extension is allowing entries from different domains, e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, or even $\mathbb{C}$. In [23], it has been shown that using complex entries enables *Quantum Multiple-valued Decision Diagrams* (QMDDs), a representation form for quantum circuits. Introducing different domains for matrix entries works comparably to multiplicative edge weights. Nevertheless, this extension regularly requires careful constructions, for different reasons: First, either one switches to transformation matrices rather than decomposition matrices to allow the usage of different multiplicative edge weights at the same decomposition level (e.g., if several nodes are marked with the same decision variable), or one requires a normalised form that has only one node per decision variable. Second, to allow canonical representations, one needs to define normalisation factor matrices that extract common factors.

Another extension is dimensionality. First, this becomes necessary as soon as more than a two-valued decision variable is introduced. Second, we expect a dimensionality extension to be helpful when representing finite state machines with the help of decision diagrams. With the results of [24], this approach seems promising for verifying sequential circuits.

V. Conclusion and Outlook

Handing over important tasks to electronic devices is only possible, when the correctness of the underlying circuits can be trusted. While simulation and emulation can give a solid idea of the correctness of hardware, only formal verification can ensure its functional correctness 100%.

With AI as rising technology, new functionalities have become standard in modern processing units, like Multiply-Accumulate functions when using neural networks. These new functionalities pose a new challenge for formal verification. While up until now, data structures that work either on the bit-level (like BDDs for adders) or on the word-level (like *BMDs for multipliers) have been used for formal verification of said circuits, it becomes necessary to design hybrid data structures that allow representing both bit- and word-level functions efficiently.

We showcased the effectiveness of hybrid data structures with the example of K*BMDs that are one example of a data structure that can be used for both bit- and word-level functions. Furthermore, we propose leveraging the Kronecker product of matrix representations of existing data structures like KFDDs, by extending their element space or dimension to enable the design of new hybrid data structures that will enable the efficient formal verification of complex functions.

References

[1] V. Pratt, "Anatomy of the pentium bug," in *Theory and Practice of Software Development (TAPSOFT)*, P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 97–107.

[2] C. Arthur, "Intel warns of $1bn cost of chip fix," https://www.theguardian.com/technology/2011/jan/31/intel-warns-cost-chip-fix, 2011.

[3] B. Chen, Y. Wang, P. Shome, C. W. Fletcher, D. Kohlbrenner, R. Paccagnella, and D. Genkin, "GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers," in *USENIX Security*, 2024.

[4] OneSpin, "Asic synthesis verification from rtl code to final netlist," https://onespin.com/products/360-ec-asic.

[5] G. Kovásznai, H. Veith, A. Fröhlich, and A. Biere, "On the complexity of symbolic verification and decision problems in bit-vector logic," in *Mathematical Foundations of Computer Science 2014*, E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 481–492.

[6] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[7] R. Drechsler and B. Becker, "Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 965–973, 1998.

[8] R. E. Bryant and Y. Chen, "Verification of arithmetic circuits with binary moment diagrams," in *Design Automation Conference (DAC)*, ser. DAC '95, New York, NY, USA, 1995, p. 535–541. [Online]. Available: https://doi.org/10.1145/217474.217583

[9] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2021, pp. 99–104. [Online]. Available: https://doi.org/10.1109/DDECS52668.2021.9417052

[10] A. Mahzoon and R. Drechsler, "Polynomial formal verification of prefix adders," in *Asian Test Symposium (ATS)*, 2021, pp. 85–90. [Online]. Available: https://doi.org/10.1109/ATS52891.2021.00027

[11] ——, "Polynomial formal verification of arithmetic circuits," *Foundations and Trends® in Electronic Design Automation*, vol. 14, no. 3, pp. 171–244, 2024. [Online]. Available: http://dx.doi.org/10.1561/1000000059

[12] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 205–213, 1991.

[13] M. Keim, R. Drechsler, B. Becker, M. Martin, and P. Molitor, "Polynomial formal verification of multipliers," *Formal Methods Syst. Des.*, vol. 22, no. 1, pp. 39–58, 2003. [Online]. Available: https://doi.org/10.1023/A:1021752130394

[14] Google, "Introduction to cloud TPU," https://cloud.google.com/tpu/docs/intro-to-tpu.

[15] R. Drechsler, B. Becker, and S. Ruppertz, "The K*BMD: A verification data structure," *Design & Test of Computers*, vol. 14, no. 2, pp. 51–59, 1997.

[16] B. Becker and R. Drechsler, "How many decomposition types do we need? [decision diagrams]," in *Proceedings of the 1995 European Conference on Design and Test*, ser. EDTC '95. USA: IEEE Computer Society, 1995, p. 438.

[17] E. M. Clarke, M. Fujita, and X. Zhao, "Multi-terminal binary decision diagrams and hybrid decision diagrams," 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:13175959

[18] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, *Formal Methods in System Design*, vol. 10, no. 2/3, p. 171–206, 1997. [Online]. Available: http://dx.doi.org/10.1023/A:1008699807402

[19] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams," in *Proceedings The European Conference on Design Automation*. IEEE Computer Society, 1992, pp. 43–44.

[20] Y.-T. Lai and S. Sastry, "Edge-valued binary decision for multi-level hierarchical verification," in *[1992] Proceedings 29th ACM/IEEE Design Automation Conference*, 1992, pp. 608–613.

[21] E. Clarke, M. Fujita, and X. Zhao, "Hybrid decision diagrams. overcoming the limitations of MTBDDs and BMDs," in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, 1995, pp. 159–163.

[22] D. E. Muller, "Application of Boolean algebra to switching circuit design and to error detection," *Transactions of the I.R.E. Professional Group on Electronic Computers*, vol. EC-3, no. 3, pp. 6–12, 1954.

[23] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2016.

[24] C. Dominik and R. Drechsler, "Polynomial formal verification of sequential circuits," in *Design, Automation & Test in Europe (DATE)*, 2024.