Taylor Expansion Diagrams: A Compact, Canonical Representation with Applications to Symbolic Verification

Maciej J. Ciesielski* Priyank Kalla* Zhihong Zeng* Bruno Rouzeyre†
*Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, USA
†Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, France

Abstract: *This paper presents a new, compact, canonical graph-based representation, called Taylor Expansion Diagrams (TEDs). It is based on a general non-binary decomposition principle using Taylor series expansion. It can be exploited to facilitate the verification of high-level (RTL) design descriptions. We present the theory behind TEDs, comment upon its canonicity property and demonstrate that the representation has linear space complexity. Its application to equivalence checking of high-level design descriptions is discussed.*

I. Introduction

Increasing size and complexity of digital designs has made essential the need to address critical verification issues at the early stages of design cycle. This requires robust, automated verification tools at higher (RT or behavioral) levels of abstraction. This paper presents a new compact, canonical, graph-based representation for algebraic expressions and Boolean functions, called *Taylor Expansion Diagram*, or *TED*. This diagram, due to its compactness and the canonicity property, can be exploited to facilitate equivalence checking of high-level (say, RTL) representations of digital designs containing arithmetic data-paths interspersed with random Boolean logic.

While formal verification systems such as [1] [2] are generally geared towards verification of control-dominated applications, verification of designs with arithmetic data-paths is mostly confined to bit-level representation [3] [4]. Abstraction of symbolic, word-level variables has been addressed in formal verification techniques, such as theorem-proving, automated decision procedures for Presburger arithmetic [5] [6], algebraic manipulation [7], term re-writing using attribute syntax trees [8], etc. Yet, the issue of verification of large designs, say, equivalence checking of RTL descriptions with complex word-level (arithmetic) and bit-level (Boolean) interaction has not been satisfactorily addressed.

Majority of verification techniques that target arithmetic-Boolean interaction in design specifications are based on generic *Word Level Decision Diagrams* (WLDDs); these are graph-based representations of functions that allow to represent functions with a Boolean domain and an integer range. Examples of WLDDs are MTBDDs [9] [10], EVBDDs [11], *BMDs [12], K*BMDs [13], FDDs[14], KFDDs [15] HDDs [16], etc. A good review of WLDDs can be found in [17]. Most existing decomposition methods for WLDDs are based on a point-wise, or *binary* decomposition of Boolean function. Different flavors of Boolean decomposition (Shannon, Davio, Reed-Muller, Kronecker, etc) lead to different *Decision Diagrams*, with or without edge weights (BDDs [18], MTBDDs [9], ADDs [10], EVBDDs [11], FDDs [14], KFDDs [15], etc). The decomposition at each node is binary and leads to exactly two terms, while the leaf nodes hold integer constants.

Binary Moment Diagram (*BMD [12]), and its derivatives (such as K*BMDs [13]), depart from such point-wise, binary decomposition, and perform a decomposition of a linear function based on its moments (constant and first moment). There are weights associated with the graph edges, which are combined multiplicatively. Several rules for manipulating edge weights are imposed on the graph to allow the graph to be canonical. Most of the arithmetic operations ($X, X+Y, X \cdot Y$, etc) have the representation that is linear in the number of variables. However, the size of X^k is polynomial in the number of bits of X, for k larger than 2. K*BMD attempts to make the BMD decomposition more efficient in terms of the graph size. This is done by applying a *set* of decompositions in a single graph, and allowing both the multiplicative and additive weights assigned to the graph edges. However, the set of restrictions imposed on the edge weights to make it canonical makes such a graph difficult to construct.

Our representation is based on an entirely different decomposition principle. It is obtained by treating the expression as a continuous, differentiable function; the decomposition is performed along the word-level, algebraic variables using *Taylor series expansion*. At each decomposition level, the expression is decomposed along one variable. The resulting *Taylor Expansion Diagram* (*TED*), is *canonical* for a fixed ordering of variables. For algebraic expressions typically encountered in RTL specifications (such as $X+Y, X \cdot Y, X^k$ for arbitrary k, etc) it is *linear* in the number of variables. Moreover, Taylor Expansion Diagrams can also be used for functions containing both algebraic and Boolean expressions, facilitating the representation of arithmetic functions with bit-nibbling and Boolean logic, typically encountered in RTL specifications.

II. Taylor Expansion Diagrams

In this representation the variables of the system are represented as real numbers (in practice they they will be implemented by integers, but no such assumption is necessary to derive the representation). Assume a regular algebra (*,+) over real numbers R, with integer coefficients. Let $f(x, y, ...)$ be a real, differentiable function corresponding to an algebraic expression F. Using Taylor series expansion *w.r.t.* variable x at

an initial point, $x_0 = 0$, the function can be represented as:

$$f(x) \quad = \quad f(0) + xf'(0) + \frac{1}{2}x^2 f"(0) + ... \qquad (1)$$

where $f'(x)$, $f''(x)$, etc, are first, second, and higher derivatives of f *w.r.t.* x. Subsequently, if those derivatives are independent of variable x, they can be further decomposed over the remaining variable using Taylor series expansion, one variable at a time. The resulting decomposition can be efficiently stored in a decomposition diagram, named *Taylor Expansion Diagram*, or *TED*, which is the subject of this paper.

A. Construction of the TED

TED is a directed acyclic graph (Φ, V, E, T), representing an algebraic expression (multi-variable polynomial). Φ is a top function representing the algebraic expression in question. V is a set of nodes, and E is a set of directed *weighted* edges connecting the nodes. T is a set of terminal nodes. The single top node (*root*) represents Φ. Its arcs point to the non-empty derivatives (children nodes) of the function, w.r.to a variable associated with the root. This decomposition is applied recursively to subsequent children nodes, resulting in the Taylor expansion diagram.

Every node v has a label (index) $var(v)$ which identifies a decomposing variable. The variables of the TED are ordered. The function at node $v \in V$ is determined by the Taylor series expansion, according to eq.(1). The internal nodes are in one-to-one correspondence with the successive derivatives of function $f(x)$evaluated at $x = 0$: $f(0), f'(0), f''(0)$, etc. The out-degree of each node v depends on the order of the polynomial function (*w.r.t.* its decomposing variable, $var(v)$), rooted at this node. The out-degree of a terminal node $v \in T$ is 0. The function of a terminal node is constant (integer).

Figure 1 shows a one-level decomposition of function f at variable x, and the notation used in the text. We shall refer to the k-th derivative of a function rooted at node v with $var(v) = x$ as a *k-child* of v: $f(x{=}0)$ is a 0-child, $f'(x{=}0)$ is a 1-child, $\frac{1}{2!}f''(x{=}0)$ is a 2-child, etc. We shall also refer to the corresponding arcs as *0-edge* (dotted), *1-edge* (solid), *2-edge* (double), etc. Notice the implicit multiplicative factors associated with each arc: $x^0 = 1$ for the 0-edge, $x^1 = x$ for the 1-edge, x^2 for the 2-edge, etc. Furthermore, the graph edges are assigned weights, computed directly from the coefficient of Taylor expansion.

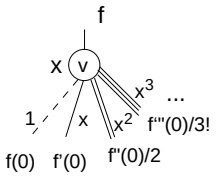

Fig. 1. Decomposition node in TED.

Figure 2 shows an example of a TED representation for the algebraic expression $(A + B)(A + 2C)$. The function represented by the TED can be computed as follows: Every path from the root node to a non-zero vertex corresponds to a non-zero term in the expression. Traversing the 2-edge from root node A and multiplying all edge-weights encountered on the paths leads to the term: $A^2 * 1 = A^2$. Similarly, following the 1-edges from A, and subsequently B, to the terminal node **1** corresponds to $A * B * 1 = AB$, and so on. Since the expressions correspond to terms of the Taylor series, all the terms are added to compute the function. Hence the diagram represents $A^2 + AB + 2AC + 2BC$.

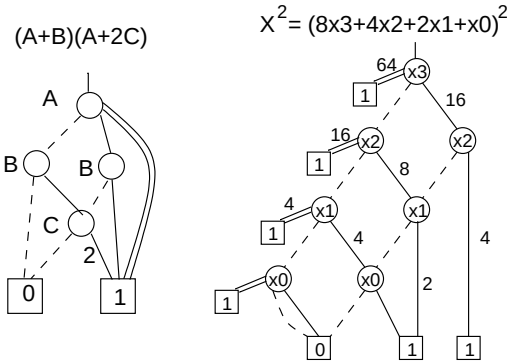

Fig. 2. Examples for expressions represented using TEDs

B. Composition Rules - the ADD and MULT Operators

Notice that the computation of the derivatives, and hence the children of f, performed recursively, is trivial for polynomial functions. Complex expressions can be composed from the simpler ones using simple composition rules and the COMPOSE operators described below. Here we will only describe the ADD and MULT operators; the subtract (SUB) operator can be obtained by a combination of ADD and MULT by -1.

The general composition process of two TEDs is similar to that of BDD's APPLY operator in the sense that it is a recursive process. However, the specific composition rules for TED diagrams are different as they must satisfy the rules of the algebra $(R, *, +)$. Furthermore, the two basic operators on TEDs, *add* $(+)$ and *mult* $(\cdot)$, are each governed by different rules.

Starting from the roots of the two TEDs, we construct the TED of the result (for a given operation) by recursively constructing all the non-zero terms of the two functions, and combining them to form the diagram for the new function. To ensure that the result is *reduced*, the REDUCE operator, described in Section III, will be applied afterwards.

Let u and v be two nodes to be composed, resulting in a new node q. Let $var(u) = x$ and $var(v) = y$ denote the decomposing variables associated with the two nodes. As explained below, $var(q)$ assumes the variable with the higher order among the two: $var(q) = x$, if $x \geq y$, and $var(q) = y$ otherwise. Let f, g be two functions rooted at nodes u, v, respectively, and h be a function rooted at the new node q. Finally, let T be a set of terminal nodes; for $v \in T$ let $val(v)$ denote the value of node v (constant).

In the following, only constant (0-) and linear (1-) edges are considered for a given node, but the analysis is equally applicable to an arbitrary number of children at each node. In constructing these basic operators, we must consider the following cases:

1. Both nodes are terminal nodes, $u, v \in T$. Then a new *terminal* node q is created as follows

ADD: $q \leftarrow (u + v)$: $val(q) = val(u) + val(v)$.

MULT: $q \leftarrow (u \cdot v)$: $val(q) = val(u) \cdot val(v)$.

2. At least one of the nodes is non-terminal; proceed according to the variable order.

(a) If the nodes have the same index then $var(q) = x$.

ADD: $q \leftarrow (u + v)$:

$$\begin{aligned} h(x) &= f(x) + g(x) \\ &= f(0) + xf'(0) + g(0) + xg'(0) \\ &= [f(0) + g(0)] + x[f'(0) + g'(0)]. \end{aligned} \quad (2)$$

That is, the corresponding k-children are paired accordingly.

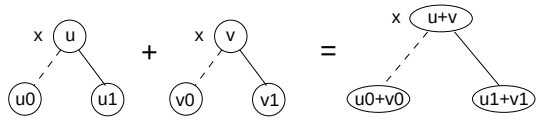

Fig. 3. Additive composition for nodes with same variable

MULT: $q \leftarrow (u \cdot v)$:

$$\begin{aligned} h(x) &= f(x) \cdot g(x) \\ &= (f(0) + xf'(0)) \cdot (g(0) + xg'(0)) \\ &= [f(0)g(0)] + x[f(0)g'(0) + f'(0)g(0)] \\ &\quad + x^2[f'(0)g'(0)]. \end{aligned} \quad (3) \quad (4)$$

Here, the 0-child of q is obtained by pairing the 0-children of u, v. Its 1-child is created as a sum of two cross products of 0- and 1-children, thus requiring an additional ADD operation. Also, an additional 2-child (representing the quadratic term) is created by pairing the 1-children of u, v.

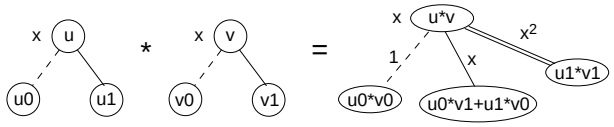

Fig. 4. Multiplicative composition for nodes with same variables

(b) If the two nodes are indexed by different variables, $var(q)$ = $max(var(u), var(v))$. Let $ord(x) > ord(y)$.

ADD: $q \leftarrow (u + v)$:

$$\begin{aligned} h(x) &= f(x) + g(y) \\ &= f(x = 0) + xf'(x = 0) + g(y) \\ &= [f(x = 0) + g(y)] + xf'(x = 0). \end{aligned} \quad (5)$$

That is, $g(y)$ of node v is added to the 0-child of node u, while the 1-child of u remains unchanged.

MULT: $q \leftarrow (u \cdot v)$:

$$\begin{aligned} h(x) &= f(x) \cdot g(y) \\ &= (f(x = 0) + xf'(x = 0)) \cdot g(y) \\ &= [f(x = 0) \cdot g(y)] + x[f'(x = 0) \cdot g(y)] \end{aligned} \quad (6)$$

Here, all children of node u are multiplied by $g(y)$ of node v.

Figure 5 illustrates the application of the COMPOSE procedure to two TEDs.

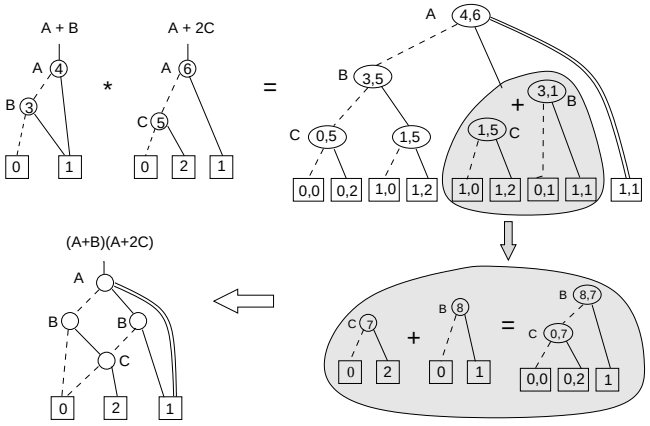

Fig. 5. Example of MULT composition: (A+B)(A+2C)

III. Properties of TED Representation

In this section we describe the construction of a reduced TED and prove its canonicity. We also discuss the complexity of the TED representation, and compare it with other known representations based on decision diagrams. The reduction of the TED representation is based on the removal of redundant nodes and merging of isomorphic subgraphs.

Definition III.1: The node is *redundant* if: 1) all of its edges are empty (point to terminal node 0); or 2) the node contains only a constant term (0-edge). In both cases, the parent of node v is routed to the 0-child of v (which in the terminal case is terminal node 0).

Definition III.2: Two TEDs are considered *isomorphic* if they match in both their structure and their attributes.

Note that since a TED contains a single root, and the children of any non-terminal vertex are distinguished, checking for isomorphism on TEDs is as easy as that for BDDs.

Definition III.3: A Taylor expansion diagram is *reduced* if it contains no redundant nodes and has no distinct vertices v and v', such that the subgraphs at v and v' are isomorphic.

We also assume that the TED is *ordered*: on all paths from root to terminal nodes the variables appear in the same order and each variable appears only once in each path.

It now remains to be shown that such constructed ordered, reduced Taylor Expansion Diagrams are canonical for a fixed variable ordering; *i.e.* every algebraic expression is represented by a *unique* Taylor expansion diagram. We state the following theorem.

Theorem 1: For any algebraic expression f of polynomial form with integer coefficients, there is a unique (up to isomorphism) ordered, reducedTaylor Expansion Diagram denoting f, and any other expression for f contains more vertices; i.e. an ordered, reduced, TED is minimal and canonical.

Proof: The proof of this theorem is conceptually straightforward, following the arguments to prove the canonicity of ROBDDs [18]. First, by construction, TED has no trivial redundancies, as all empty edges with zero weight are eliminated. Second, for polynomial functions, the Taylor series expansion at a given point (x_0=0) is finite. It remains to be shown that the individual Taylor expansion terms, evaluated recursively, are unique for a given subfunction (node). This is obvious, con-

sidering the unique representation of the recursively computed derivatives. Finally, the sharing of identical terms across different decomposition levels is captured by the reduction operation; hence minimality. From the above discussions it follows that an ordered and reduced TED is a compact, canonical and efficient representation that can be exploited for RTL equivalence checking purposes.

A. Complexity of the TED Representation

For linear expressions, the space complexity of TED is the same as *BMD, i.e., linear. This is also true for multipliers, $X \cdot Y$, regardless of the size of the bit vector representation, since the product of two or more *different* bit vectors leads to polynomials that are linear in the number of the bit variables. For polynomials of degree $k \geq 2$ the size of *BMD grows $O(n^k)$ with the number n of bits. For K*BMD the representation also becomes nonlinear ($O(n^{k-1})$) for polynomials of degree 3 and higher, while TED remains linear in the number of bits, regardless of the degree k. The TED representation for X^2, where $X = \{x_3, \ldots, x_0\}$ is shown in Fig. 2. We state the following theorem:

Theorem 2: Let variable X be represented as a bit vector of width n: $X = \sum_{i=0}^{n-1} 2^i x_i$. The number of TED nodes needed to represent X^k is $k(n-1)+1$.

Proof: We shall first illustrate it for the quadratic case $k = 2$. Let W_n be an n-bit representation of X: $X = W_n = \sum_{i=0}^{n-1} 2^i x_i = 2^{(n-1)} x_{n-1} + W_{n-1}$ where $W_{n-1} = \sum_{i=0}^{n-2} 2^i x_i$ is the part of X containing the lower (n-1) bits. With that, $W_n^2 = (2^{n-1} x_{n-1} + W_{n-1})^2 = 2^{2(n-1)} x_{n-1}^2 + 2^n x_{n-1} W_{n-1} + W_{n-1}^2$. Furthermore, let $W_{n-1} = (2^{n-2} x_{n-2} + W_{n-2})$, and $W_{n-1}^2 = (2^{2(n-2)} x_{n-2}^2 + 2^{n-1} x_{n-2} W_{n-2} + W_{n-2}^2)$.

Notice that the "constant" term (0-edge) of W_{n-1} w.r.to variable x_{n-2} contains the term W_{n-2}, while the linear term (1-edge) of W_{n-1}^2 contains $2^{n-1} W_{n-2}$. This means that the term W_{n-2} can be *shared* at this decomposition level by two different parents.As a result, there are exactly two non-constant terms, W_{n-2} and W_{n-2}^2 at this level.

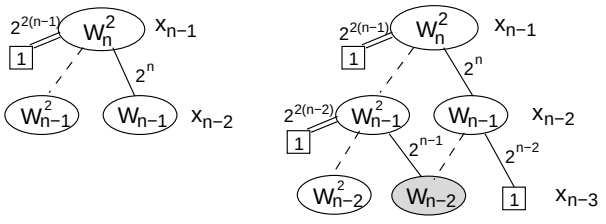

Fig. 6. Construction of TED for X^2 with n bits

In general, at any level l, associated with variable x_{n-l}, the expansion of terms W_{n-l}^2 and W_{n-l} will create *exactly two* different non-constant terms, one representing W_{n-l-1}^2 and the other W_{n-l-1}; plus a constant term 2^{n-l}. The term W_{n-l-l} will be shared, with different multiplicative constants, by W_{n-l}^2 and W_{n-l}.

This reasoning can be easily generalized to arbitrary integer degree k; at each level there will always be exactly k different non-constant terms. Since on the top variable (x_{n-1}) level there is only one node (the root), and there are exactly k non-constant nodes at each of the remaining $(n-1)$ levels, the total number of nodes is equal to $k(n-1)+1$. □

IV. Representing Boolean functions and Boolean-Arithmetic interaction

In this section we propose enhancements to the TED representation so that it can properly represent Boolean logic and interface it with arithmetic datapath units of the design. Recall that TEDs have been devised for functions with integer range and integer domain. While restricting integer range to {0,1} is trivial, such a restriction for the domain of an integer function requires a modification of the corresponding algebraic function to assume Boolean values. This is described below.

We define TED operators for Boolean logic, where both the range and domain are Boolean. Correct derivation of the operations must be based on the rules of regular algebra $R(+,*)$, and not on Boolean algebra; furthermore the individual terms must be disjoint. Analogous to their *BMD counterparts, the following formulas can be derived for basic TED operators for Boolean logic (or BTED operators): $NOT(x) = x' = (1-x); AND(x,y) = x \wedge y = x * y; OR(x,y) = x \vee y = x + y - x * y; XOR(x,y) = x \oplus y = x + y - 2 * x * y$. Any circuit containing purely Boolean logic can be easily represented using BTEDs.

A. Interfacing Arithmetic and Boolean logic

We already know that a purely Boolean circuit represented with TED (BTED) will always correctly compute its *Boolean* logic value. It now remains to show that modeling the interaction of an *arithmetic* or word-level variable with one or more Boolean variables gives a correct representation of the resulting arithmetic, integer-valued function.

We claim that the only case that remains to be considered is the occurrence of Boolean variables x^k, with $k \geq 2$ (*e.g.* $F = G + x + x^2, G =$ algebraic expression, $x \in \{0,1\}$), as they are the only ones that can possibly lead to incorrect computation of arithmetic values. This is because the functions are arithmetic (integer-valued), their computation is based on $(R,+,*)$ algebra, while the variables can be both the *word-level*, with an integer range, and *Boolean*, with range {0,1}.

The arithmetic ADD function will never create incorrect results over Boolean and arithmetic variables. For example, $F = G = x + x = G + 2x$ will always produce the correct results for $x \in R$ or for $x \in \{0,1\}$. However the MULT function may create powers of Boolean variables, x^k, thus potentially leading to problems in evaluating the function. Moreover, this may only happen for variables that are related through their expansion; for example, when multiplying a word-level variable $X = \sum_{i=0}^{n-1} 2^i x_i$ and a Boolean variable x_l, for some $l \leq n-1$. Notice that for $x \in \{0,1\}, x = x^2 = \cdots = x^k$, for $k \geq 2$. Hence, in this case we may replace all the occurrences of x^k with x, without changing the value of the function. This allows us to correctly evaluate the value of the function without really changing the Boolean nature of the variable x. This modification allows to model the Boolean-algebraic interface of variables for TED representation as shown below: MULT: $q \leftarrow (u \cdot v)$:

$$h(x) = f(x) \cdot g(x) = (f(0) + xf'(0)) \cdot (g(0) + xg'(0)) \quad (7)$$

$$= [f(0)g(0)] + x[f(0)g'(0) + f'(0)g(0) + f'(0)g'(0)], \quad (8)$$

as obtained by replacing x^2 term by x. It can be shown that the modified BTEDs are canonical and can be directly used for equivalence checking of RTL designs. In the next subsection, we describe how we can use the above modifications to TEDs to represent complex RTL descriptions of digital designs for equivalence checking purposes.

B. Application to RTL Verification

Consider the two circuits shown in Fig. 7. They correspond to two different high-level (RTL) implementation of equivalent designs that need to be verified for equivalence. Their corresponding equivalent expressions are: $F_1 = s_1(A+B)(A-B) + (1-s_1)D$ and $F_2 = s_2 D + (1-s_2)(A^2 - B^2)$, where A and B are n-bit arithmetic variables, and a_k, b_k are Boolean variables (bits), related to A, B. To prove the equivalence of the two arithmetic expressions, the TED representation is constructed for each design. First, the arithmetic variables A and B are partially expanded as follows, to facilitate the "bit nibbling" with a_k, b_k: $A = 2^{(k+1)}A_{hi} + 2^k a_k + A_{lo}; B = 2^{(k+1)}B_{hi} + 2^k b_k + B_{lo}$. Here, A_{hi}, B_{hi}, and A_{lo}, B_{lo} are arithmetic variables (not expanded into bits) of word-length $(n-k-1)$ and k, respectively, and a_k, b_k are single bits. The TEDs are built for $(A-B)(A+B)$ from its components. The TEDs for the Boolean logic s_1, s_2 are then constructed, and composed with the corresponding arithmetic functions at the inputs of the MUX.

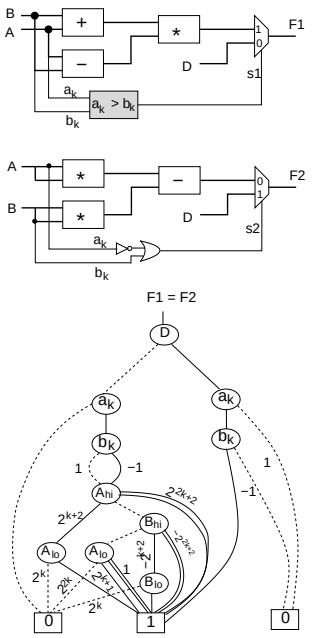

Fig. 7. Equivalence check for two RTL designs using TED

The bottom part of Figure 7 shows the isomorphic TED for both of these designs, demonstrating that they are equivalent.

V. CONCLUSIONS

This paper has presented Taylor Expansion Diagrams: a new symbolic, compact, canonical representation that can efficiently represent not only both arithmetic and Boolean functions, but also model the interaction between them. This representation is based on a new, non-binary decomposition, that treats the arithmetic expression as a continuous, differentiable function over symbolic (algebraic) variables and applies the Taylor series expansion. We presented the theory behind the TED, proved that it has linear space complexity and and showed that when reduced and ordered, TED is canonical. We also presented a modification that allows TED to properly handle a combination of arithmetic circuits and Boolean logic, while preserving the canonicity property. The proposed representation can be exploited for equivalence checking of RTL specifications containing complex interaction between arithmetic and Boolean parts of the design.

We are currently implementing the TED package to be used by verification tools for equivalence checking. We wish to evaluate its applicability to realistic designs that contain arithmetic circuits with Boolean logic. While we do not have results to demonstrate the effectiveness of TEDs for verification, we believe that this new theory will significantly advance the state-of-the-art in the symbolic verification arena.

REFERENCES

[1] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vencentelli, F. Somenzi, A. Aziz, S-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, G. Shiple, S. Swamy, and T. Villa, "VIS: A System for Verification and Synthesis", *in Computer aided Verification*, 1996.

[2] K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[3] D. Cyrluk, O. Moller, and H. Ruess, "An Efficient Procedure for the Theory of Fixed-Size Bitvectors", *in LCNS, CAV*, vol. 1254, 1997.

[4] C. W. Barlett, D. L. Dill, and J. R. Levitt, "A Decision Procedure for bit-Vector Arithmetic", *in DAC*, June 1998.

[5] H. Enderton, *A mathematical introduction to logic*, Academic Press New York, 1972.

[6] T. Bultan and et al, "Verifying systems with integer constraints and boolean predicates: a composite approach", *in In Proc. Int'l. Symp. on Software Testing and Analysis*, 1998.

[7] S. Devadas, K. Keutzer, and A. Krishnakumar, "Design verification and reachability analysis using algebraic manipulation", *in Proc. ICCD*, 91.

[8] Z. Zhou and W. Burleson, "Equivalence Checking of Datapaths Based on Canonical Arithmetic Expressions", *in DAC*, 95.

[9] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping", *in DAC*, pp. 54–60, 93.

[10] I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and their Applications", *in ICCAD*, pp. 188–191, Nov. 93.

[11] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "FGILP: An ILP Solver based on Function Graphs", *in ICCAD*, pp. 685–689, 93.

[12] R. E. Bryant and Y-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams", *in DAC*, 95.

[13] R. Dreschler, B. Becker, and S. Ruppertz, "The K*BMD: A Verification Data Structure", *IEEE Design & Test*, pp. 51–59, 1997.

[14] U. Kebschull, E. Schubert, and W. Rosentiel, "Multilevel Logic Synthesis based on Functional Decision Diagrams", *in EDAC*, pp. 43–47, 92.

[15] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Order Kronecker Function Decision Diagrams", *in DAC*, pp. 415–419, 1994.

[16] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid Decision Diagrams - Overcoming the Limitation of MTBDDs and BMDs", *in ICCAD*, 95.

[17] S. Horeth and Drechsler, "Formal Verification of Word-Level Specifications", *in DATE*, pp. 52–58, 1999.

[18] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, August 1986.