Circuit-based Evaluation of the Arithmetic Transform of Boolean Functions

René Krenz[1*] Elena Dubrova[1] Andreas Kuehlmann[2]

[1] Royal Institute of Technology, IMIT/KTH, 164 46 Kista, Sweden
[2] Cadence Berkeley Labs, Berkeley, CA 94704, USA

Abstract

In this paper we present a fast algorithm for evaluating the arithmetic transform of a Boolean function based on its circuit representation. The arithmetic transform has multiple applications in CAD, including the computation of signal probabilities and switching activities of circuit nets and the mapping of Boolean functions onto probabilistic hash values. Previous algorithms for evaluating the arithmetic transform required an orthogonal, non-redundant representation of the function to be transformed in form of a disjoint function cover or a single BDD. We present a new algorithm that partitions the evaluation based on the dominator relations of the circuit graph. Similar to the application of cut-points in combinational equivalence checking, the dominators are used to progressively simplify intermediate evaluation steps. As a result, the presented algorithm can handle larger circuits than previously possible. An extensive set of experiments on benchmark and industrial circuits demonstrate the effectiveness of our approach.

1 Introduction

The arithmetic transform [1] has multiple applications in CAD, including the computation of signal probabilities for test generation, switching activities for power and noise analysis, and function-specific hash values for probabilistic verification.

The *signal probability* of a net in a combinational circuit is the probability that a randomly generated input vector will produce the value one on this net [2]. Efficient signal probability analysis allows to improve the coverage of test generation for biased random simulation. The average *switching activity* in a combinational circuit is the probability of its net values to change from 0 to 1 or vice versa. It correlates directly with the average power dissipation of the circuit [3]; thus its analysis is useful for guiding logic optimization methods to target low-power dissipation. A third application of the arithmetic transform *hashes Boolean functions* onto single integer values and is used to probabilistically compare them [4].

Suppose the onset of a Boolean function f with input variables $\{x_1,\dots,x_n\}$ is given as a set of minterms $\{m_1,m_2,\dots,m_k\}$, where each $m_j=(x_1\oplus\alpha_{1j})\wedge\dots\wedge(x_n\oplus\alpha_{nj}),\alpha_{ij}\in\{0,1\}$. For a given input assignment $\{a_1,\dots,a_n\}$ of values chosen from a field F, the value of the arithmetic transform $A[f]$ is defined by:

$$A[f](a_1,\dots,a_n)=\sum_{j=0}^{k-1}\Big[\prod_{\forall i.\alpha_{ij}=0}(a_i)\;\cdot\prod_{\forall i.\alpha_{ij}=1}(1-a_i)\Big] \quad (1)$$

where $\cdot$, $+$, $-$ are operations over the field F.

If the individual a_i's are the independent switching activities at the inputs, with values between 0 and 1 from a field of real numbers, then $A[f]$ represents the switching activity of a net implementing f. Similarly, if a_i represents the uncorrelated probability that input x_i is one, then $A[f]$ is the probability that the corresponding net is one. Finally, if the a_i's are independently and uniformly at random assigned values from an integer field Z_p, p - prime, then $A[f]$ is the hash value for f.

Note that in the given applications, we are only interested in the evaluation of $A[f]$ for specific values of the a_i's. For this we do not necessarily need to compute the complete transform, for example, as a monolithic symbolic expression. The challenge for an effective practical application is to compute the evaluation efficiently in the common case where f is given as a circuit without having to build a normal form as used in definition (1).

A decomposed evaluation based on topologically processing the circuit gates from inputs to outputs using the transforms of the gate functions (e.g. $A[f\wedge g]=A[f]\cdot A[g], A[\neg f]=1-A[f]$) generally produces incorrect results. For example, if the support sets of two sub-functions f and g overlap, then $A[f\wedge g]\neq A[f]\cdot A[g]$. Overlapping support sets cause higher-order exponents, which need to be suppressed to get correct results [2]. To avoid this difficulty, previous works on arithmetic transforms require a non-redundant, orthogonal representation of the function to be transformed.

In this paper we present a new algorithm that partitions the evaluation of the arithmetic transform using the dominator relations of the circuit graph. Proceeding in topological order, the algorithm computes a symbolic expression for the transform and successively simplifies it at dominator points by partially evaluating its terms. We exploit the fact that the dominators in a circuit graph provide the earliest points during topological processing at which expressions can be simplified and that the dominator relation gives the maximum number of terms that can be evaluated at these points. Interestingly, about 50% of practical circuit graphs contain a significant number of internal dominator vertices which warrants an efficient performance of the presented algorithm. Corresponding statistics and results are give in Section 7.

Our main application of the presented work is focused on probabilistic verification. If the arithmetic transform is based on an

[*] The author carried out this work in part while visiting Cadence Berkeley Labs.

integer field, its evaluation returns the hash value of a Boolean function for a given input assignment. For a field of large size p (e.g. $p = 2^{32} - 5$, using word-wide processor instructions), the hash value strongly discriminate Boolean functions; two circuits with different functions of n inputs are classified as equivalent with the low probability of $\approx \frac{n}{p}$ [4], whereas functionally equivalent circuits are guaranteed to be identified as equal. Possible applications include refutation of correctness in equivalence checking [5, Chapter 13] and bounded property checking [6], cutpoint guessing [7], detecting functional corresponding registers [8], and others. A particularly notable method that is related to the presented work is given in [9] where cutpoints are used to progressively abstract a function representation by quantification. In their work the authors did not offer a systematic manner to identify good cutpoints for the abstraction. Although used differently, the dominator-based method described here could complement their work by providing the exact points where abstraction is applicable.

A second areas that can benefit from the presented method is power analysis and optimization. Here an efficient computation of the circuit's switching activity is crucial for including the power analysis in the inner loop of the optimization flow. Besides the temporal dependencies between signal values, accurate switching analysis must also take into account their functional correlation which can be modeled by the arithmetic transform using a field of real numbers between 0 and 1 [10]. A third possible application of a fast algorithm for evaluating arithmetic transforms uses them to compute signal probabilities for measuring and controlling the coverage in testing or biased random simulation [11]. Here the aim is to tune the individual input signal probabilities for increasing particular coverage goals.

2 Previous Work

The arithmetic transform has been researched and rediscovered in many works. The pioneering publications are from Aiken [12] and Komamiya [13] where the transformation matrix for the arithmetic transform was introduced. Later, several techniques for computing the transform were presented using: truth tables [14], disjoint sum-of-products [15], ROBDDs [16], and various derivatives of decision diagrams (see [1] and [17, Chapter 3.5] for an overview). The disadvantage of these methods is that all of them use a representation based on an explicit or implicit disjoint function cover with its typical excessive memory requirements.

It is interesting to observe that similar algorithms were independently discovered and applied in the testing and low-power design community. Parker and McCluskey [2] developed two algorithms for deriving the output signal probability for a combinational circuit in terms of a set of input probabilities. Their first algorithm requires a canonical sum-of-product form for a Boolean functions. The second one works directly on a circuit description and uses higher-order exponent suppression to correct erroneous terms introduced by re-converging paths. An algorithm for estimating the average switching activity of combinational logic circuits based on disjoint sum-of-products was proposed in [10]. Decision diagram-based algorithms were used in [18, 3, 19]. All these techniques are similar to the ones mentioned before and suffer from the same memory complexity.

One option that was extensively studied in the testing community, is trading the quality of the result for speed. In [11], the cutting algorithm was proposed which computes the signal probability within certain bounds. This is done by cutting re-convergent fanout branches and turning the circuit into a tree for which the complexity of computing signal probabilities is linear [2]. Improved algorithms for computing bounds on signal probabilities are proposed in [20, 21, 22, 23]. In [24] a Boolean approximation method is proposed to avoid building a global BDD. It uses the first term of a Taylor series expansion to compute the signal probabilities. As for the previous methods, the accuracy of the results is compromised for lowering the computational complexity.

In [25] a method based on "supergates" is proposed. The output net of a supergate is required to dominate all the primary inputs in its support set. Thus, all re-converging paths are completely enclosed within supergates. The switching probability is first computed for all supergates and then composed for the entire circuit. This idea is further extended in [26] by introducing the concept of "active nodes" at which two (or more) re-convergent paths begin. These nodes are kept active until the paths meet at a dominator node where the corresponding expressions are simplified. This work is the closest to the approach presented in this paper. The main difference is that the algorithm in [26] is path-based and thus can handle only small regions of re-converging paths. For larger regions the results are estimated which is not acceptable for many applications, e.g. functional hashing.

In the verification community, arithmetic transforms were used by Jain et al in [27, 4] for comparing Boolean function probabilistically. They suggest the concept of *Seminumeric Decision Diagrams* to decompose a function into disjoint subfunctions. The subfunctions are then hashed and the obtained results are used to compute the hash code for the entire function. Seminumeric Decision Diagrams allow integer-valued terminal nodes, integer-valued weights on edges and functional nodes for field operations of addition, subtraction and multiplication. Although its is shown that Seminumeric Decision Diagrams can handle some complex circuits, they do not offer a systematic manner to analyze and exploit structural circuit properties such as the dominator-based algorithm presented in this paper.

3 Preliminaries

Unless otherwise specified, throughout the paper, we denote by f, g Boolean functions of type $\{0,1\}^n \to \{0,1\}$ and by $A[f]$, $A[g]$ arithmetic functions over a field F, of type $F^n \to F$. We use the symbols $\vee$, $\wedge$, and $\neg$ for Boolean operations AND, OR, and NOT, respectively and denote by $\cdot$, $+$, and $-$ the arithmetic operation over the field F. The indexed variables $x_1, x_2, \ldots$ or $x_a, x_b, \ldots$ are used for both, Boolean and arithmetic variables. For example, $x_1 \wedge x_2$ stands for a Boolean expression representing an AND of two Boolean variables; $x_1 \cdot x_2$ stands for an arithmetic expression (polynomial) representing a multiplication of two arithmetic variables.

Every arithmetic function $A[f]$ can be decomposed with respect to a variable x_i, $i \in \{1, \ldots, n\}$, in the following manner [27]:

$$A[f] = (1 - x_i) \cdot A[f|_{x_i=0}] + x_i \cdot A[f|_{x_i=1}]. \qquad (2)$$

It is easy to prove (by induction on n) that if the decomposition (2)

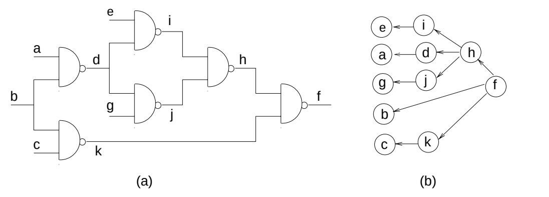

Figure 1: Example: (a) circuit diagram; (b) corresponding dominator tree.

is successively applied for all input variables we get the following normal form for $A[f]$:

$$A[f] = \sum_{j=0}^{2^n-1} f_j \cdot [\prod_{\forall i. j_i=1} (x_i) \cdot \prod_{\forall i. j_i=0} (1-x_i)] \tag{3}$$

where $f_j = f(j_1, j_2, \ldots, j_n)$ with $(j_1 j_2 \ldots j_n)$ being the binary expansion of j. Clearly, (3) reduces to (1) for $(x_1, \ldots, x_n) = (a_1, \ldots, a_n)$.

4 Illustrative Example

Our goal is to evaluate $A[f]$ for a given input assignment $(a_1, \ldots, a_n) \in F^n$. A brute-force algorithm to compute $A[f]$ directly from the circuit description first builds the complete arithmetic expression for $A[f]$ based on symbolic variables $x_1, x_2, \ldots, x_n$ for the inputs. Next, the higher-order exponents $x_i^{<j>}$, $j \geq 2$, are suppressed and then all symbolic variables in $A[f]$ are substituted by their corresponding values $a_1, a_2, \ldots, a_n \in F$. Clearly, such an algorithm is of exponential complexity in n. For example a cascade of two-input OR gates implementing an n-input OR function would produce $2^n - 1$ terms.

Our approach is to avoid computing the complete expression for $A[f]$. For this we can use the known fact that in a tree-like circuit structure with no re-convergent fanouts all signals are independent and, for any gate in the circuit, the support sets of the input functions are disjoint [2]. As a consequence, no higher-order coefficients are produced and the value for $A[f]$ can be computed directly, by starting from the input values $a_1, a_2, \ldots, a_n \in F$ and instantaneously evaluating the expression at each gate.

Usually however, circuits do have re-convergent paths. The basic idea of our algorithm is to distinguish between the nets which have re-converging structures and the ones which do not. The expressions feeding nets that do not re-converge can be evaluated immediately. However, for nets that have re-convergences this processing must be postponed until full convergence is achieved.

A key observation that led to our algorithm is that a re-converging structure always starts with a multi-fanout net and ends at a dominator net. A net v dominates another net w in the circuit cone if every path from w to the output contains v [28]. As an example, consider the circuit in Figure 1(a). In this circuit, d dominates $\{a\}$ (but not $\{b\}$), h dominates $\{a, d, e, g, i, j\}$, and f dominates all circuit nets. The dominator tree, reflecting the non-transitive dominator relations, is shown in Figure 1(b).

The idea of the presented algorithm is to process the circuit in topological order from inputs toward outputs and to successively compute the arithmetic transform $A[f_v]$ for each net v. Whenever a re-converging path begins, a temporal symbolic variable is introduced which is then eliminated at the end of the re-converging structure by partially evaluating $A[f_v]$.

The algorithm starts by assigning the input field values to the primary input nets ($A[f_v] = a_v$). For the given example, the values for primary inputs a, c, e and g can directly be used in the following gate computations, because their fanout is one and thus do not cause re-converging paths. However, the transform for b must be kept as a symbolic variable, say x_b, until its dominator is reached. Then higher-order exponents can be suppressed and x_b is substituted by its value.

To further illustrate the interaction between variable introduction and its elimination, consider net d. It is a starting point of a re-converging fanout which ends at its dominator h. The connection from b to the gate driving d causes higher-order exponents of x_b at h. However, it would not be valid to only keep x_b symbolic and suppress its higher-order exponents at h. For example, suppose $F = Z_p$, p-prime, and values 3, 5, and 1 are assigned to nets a, e and g, respectively. The evaluation results in $A[f_d] = 1 - 3x_b$, $A[f_i] = 1 - 5(1 - 3x_b) = -4 + 15x_b$, $A[f_j] = 1 - 1(1 - 3x_b) = 3x_b$, and finally $A[f_h] = 1 - (-4 + 15x_b)(3x_b) = 1 + 12x_b - 45x_b^2 = 1 - 33x_b$. This result is incorrect since the proper value for h is $A[f_h] = 1 - 3x_b$.

The reason for this problem is that the primary input variable x_a has been substituted too early. As a result, the higher-order powers of its value caused by the multiple fanout at d were not properly suppressed at h. To get the correct result, a temporary variable must be introduced for net d which is then substituted by its expression $A[f_d] = 1 - 3x_b$ at net h. Then the higher-order exponents are suppressed correctly and, furthermore, the correlation with the value of net b is preserved. Let x_d be the variable introduced at d, then we get $A[f_i] = 1 - 5x_d$, $A[f_j] = 1 - x_d$, and $A[f_h] = 1 - (1 - 5x_d)(1 - x_d) = 6x_d - 5x_d^2 = x_d$. Replacing x_d by $A[f_d] = 1 - 3x_b$, we obtain the correct expression $A[f_h] = 1 - 3x_b$.

To complete the computation of $A[f_f]$, suppose the values 2 and 4 are assigned to the primary inputs c and b, respectively. From $A[f_k] = 1 - 2x_b$ and $A[f_h] = 1 - 3x_b$, we get $A[f_f] = 1 - (1 - 2x_b)(1 - 3x_b) = x_b$. By replacing x_b with $A[f_b] = 4$, we obtain $A[f_f] = 4$, which is the correct value for the input assignment $(x_a, x_b, x_c, x_e, x_g) = (3, 4, 2, 5, 1)$.

The algorithm TRANSFORM presented in the next section is based on the intuitive idea described above.

5 Transformation Algorithm

In this section we describe a new algorithm that partitions the evaluation process of the arithmetic transform using the dominator relations of the circuit graph [29, 28, 30].

Let $C = (V, E, root)$ denote a single-output circuit graph where the set of vertices V represents the primary inputs and gates. A particular vertex $root \in V$ is marked as the circuit output. The set of edges $E \subseteq G \times G$ represents the nets connecting the gates.

A vertex v *dominates* another vertex $w \neq v$ in C if every path from w to $root$ contains v. Vertex v is the *immediate dominator* of w, denoted $v = idom(w)$, if v dominates w and every other dominator of w dominates v. It was proved in [29, 30] that every vertex v in C except $root$ has a unique immediate dominator.

The edges $\{(idom(w),w) \mid w \in V - \{root\}\}$ form a directed tree rooted at *root*, which is called the *dominator tree* of C. The dominator children $Doms(v) \subset V$ of vertex v is the set of vertices having v as immediate dominator, i.e., $Doms(v) = \{u \mid idom(u) = v\}$. We construct a *reduced dominator tree* by removing all subtrees from the dominator tree that do not have primary inputs as leave vertices. In other words, v belongs to the set of vertices $D_R \subseteq V$ of the reduced dominator tree if:

1. v is a primary input or
2. $\exists u \in D_R$ such that $v = idom(u)$.

Figure 2 shows the pseudo-code of the algorithm to evaluate the arithmetic transform. For the computation we need two labels for each vertex $v \in V$: (1) $A[v]$ stores the arithmetic expression of the vertex, and (2) $x[v]$ represents the expression to be used for the evaluation at all fanout vertices of v. If v is the beginning of a new re-converging structure, $x[v]$ is initialized with a temporary symbolic variable, otherwise $x[v]$ is set to $A[v]$. FUNCTION (v,X) is outlined for the basic functions AND and NOT. The evaluation of other functions can be easily composed from those.

```
algorithm TRANSFORM $(V,E,root,a_1,\ldots,a_n)$
  $D_R, Doms$ = DOMINATOR $(V,E,root)$
  for each $v \in V$ in topological order do
    if $v \in Inputs$ then
      $A[v] = a_v$
    else
      $A[v] = $ FUNCTION$(v,X_v)$,  $X_v = \{x_u \mid u \in Fanin(v)\}$
    if $v \in D_R$ then
      for each $u \in Doms(v)$ in reverse topological order do
        if $x[u] \neq A[u]$ then
          $A[v] = $ COMPOSE$(A[v],A[u],x[u])$
    if $v \in D_R \wedge |Fanout(v)| > 1$ then
      $x[v] = $ CREATE_FRESH_VARIABLE
    else
      $x[v] = A[v]$
  return $A[root]$;
end

algorithm FUNCTION $(v,x_1,x_2,\ldots)$
  /* rules are only shown for AND and NOT functions */
  if $v$ is AND then return $result = \prod_i x_i$
  if $v$ is NOT then return $result = 1 - x_1$
end

algorithm COMPOSE $(A[f],A[g],x)$
  return $result = A[f|_{x=0}] + (A[f|_{x=1}] - A[f|_{x=0}]) \cdot A[g]$
end
```

Figure 2: Pseudo-code of the algorithm TRANSFORM to evaluate the arithmetic transform $A[f]$.

The algorithm processes the circuit from the inputs toward the output in topological order. At the primary inputs the $A[v]$ are initialized by their corresponding values $a_v \in F$. Then, at each vertex, the new function is computed based on the x values of its fanin vertices. If vertex v is part of the reduced dominator tree, the variables $x[u]$ of the dominated vertices u are substituted in $A[v]$ by their corresponding values $A[u]$ using the function COMPOSE . This must be done in reverse topological order, otherwise dependencies between the $A[u]$ expressions may not get resolved properly.

If a vertex is part of the reduced dominator tree and has multiple fanouts, an auxiliary variable is introduced for $x[v]$ which will be substituted at vertex $idom(v)$. For reduced dominator tree vertices with single fanout, these two steps are omitted because in that case $idom(v)$ is the only successor of v.

6 Implementation Details

For computing the dominator tree in a circuit graph we implemented the Lengauer-Tarjan [31] algorithm which works efficiently for large circuits. We use multi-terminal BDDs (MTBDDs) [32] (also called ADDs [33]) to represent and manipulate the arithmetic expressions $A[f]$. The ordering and reduction rules for MTBDDss are identical to the ones for BDDs. The arithmetic function $A[f_v] : F^n \to F$ associated with a MTBDD with root v is defined recursively as follows:

1. If v is a terminal node with value $a_i \in F$, then $A[f_v] = a_i$.
2. If v is a non-terminal branching node with $index(v) = i$, then $A[f_v]$ is the function:

$$A[f_v] = (1 - x_i) \cdot A[f|_{x_i=0}] + x_i \cdot A[f|_{x_i=1}]$$

   where ”$+$”,”$-$” and ”$\cdot$” denote the operations over F.

We have proved that the operations on arithmetic functions can be performed by equivalent operations on MTBDDs, i.e. $A[f] + A[g]$ and $A[f] \cdot A[g]$ can be implemented by the MTBDD ADD and MULTIPLY operation. The compose operation of a variable x_i of $A[f]$ by function $A[g]$ is implemented as $A[f]|_{x_i=A[g]} = A[f|_{x_i=0}] + (A[f|_{x_i=1}] - A[f|_{x_i=0}]) \cdot A[g]$. The correctness follows directly from (2).

The presented algorithm can be applied to any field F. In our current implementation, we choose F to be a field of integers Z_p, p–prime, because our work is primarily targeted at probabilistic verification methods. However, it can easily be adopted to compute signal probabilities or average switching activities of logic circuits by changing Z_p to the field of real numbers, where the input variables are assigned values between 0 and 1.

7 Experiments

In this section we provide a set of experimental results for evaluating the arithmetic transform based on the presented algorithm. All experiments were performed on a PC with a 1.4 GHz Pentium4 CPU and 1 GByte main memory, running Linux RedHat 7.2. The runtime and memory consumption of algorithm TRANSFORM was compared to the performance of our own implementation of a BDD-based evaluation scheme similar to [34, 18, 3] based on building a global BDD. The CUDD-package [35] version 2.3.1 was used for the BDD and MTBDD manipulations. We have modified the MTBDD part to support integer field operations and added the COMPOSE operator described in the previous section.

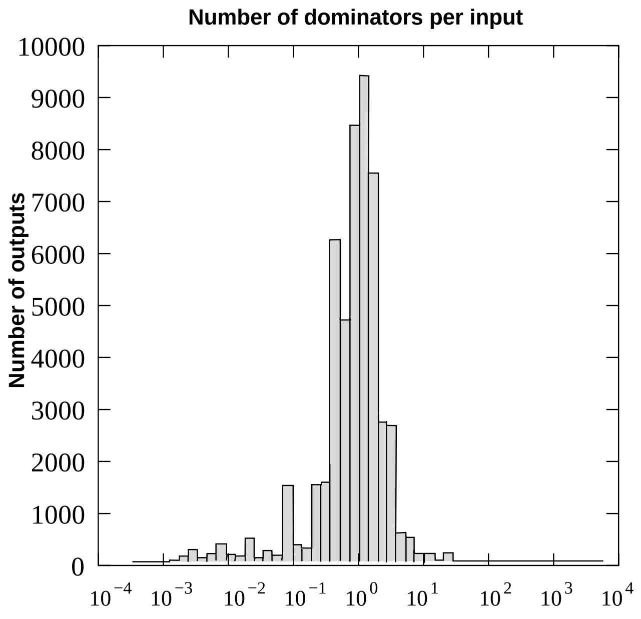

Figure 3: Histogram of the number of internal dominators per input.

For the comparison we used 253 circuits from standard benchmark and industrial circuits which comprise a total of 99,618 outputs. 48,047 of these outputs (48%) have internal dominator nets which can be used in the presented algorithm. Figure 3 shows the number of internal dominators relatively to the number of inputs for these outputs. Predominantly the amount of dominators per input lies between 5 and 0.5. This statistic demonstrates that practical circuits contain a significant amount of internal dominators which can be exploited by the presented approach.

To compare the performance of the dominator-based algorithm with the BDD-based algorithm, we evaluated the arithmetic transforms for random input assignment for all outputs that have at least one internal dominator (48,047). For both algorithms we applied a MTBDD (BDD) node limitation of 1 million alive DD nodes per output. The dominator-based and BDD-based algorithm exceeded that limit in 1,538 and 1,630 cases, respectively.

Figure 4 compares the memory usage of both algorithms in terms of the maximal number of BDD/MTBDD nodes. Each entry in the diagram represents one output and gives the maximum number of alive DD nodes for each approach. For the majority of the test cases the new algorithm requires a significantly smaller amount of nodes. However, in a number of cases the dominator-based algorithm requires more nodes. This can be explained by the fact that the BDD representation can take advantage of complemented edges whereas MTBDDs do not provide that mechanism.

Figure 5 summarizes the runtime comparison for both algorithms. Here the difference between the dominator-based and BDD-based algorithm is not as significant as for the memory comparison. This can be explained with the additional computing effort needed by the current implementation of the COMPOSE operations which is particularly notable for larger circuits. Part of our future work is to implement this operation more efficiently for eliminating this overhead.

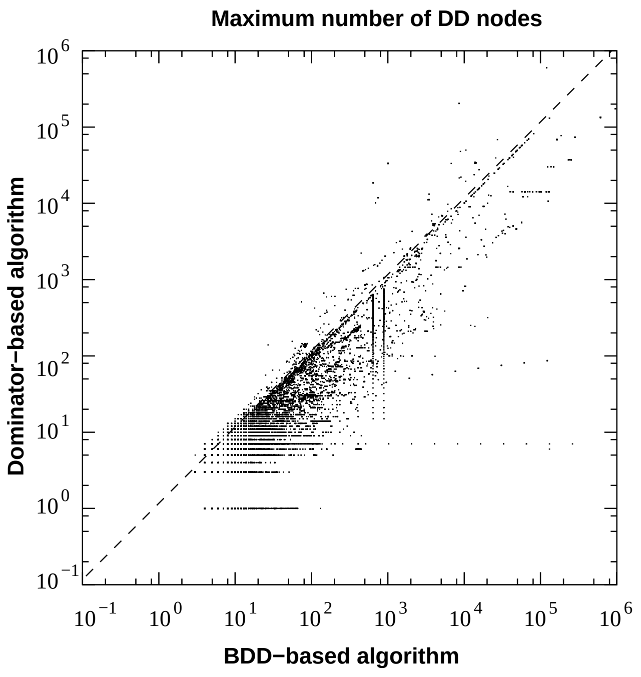

Figure 4: Comparison of the memory usage based on the maximum number of DD nodes to process an output.

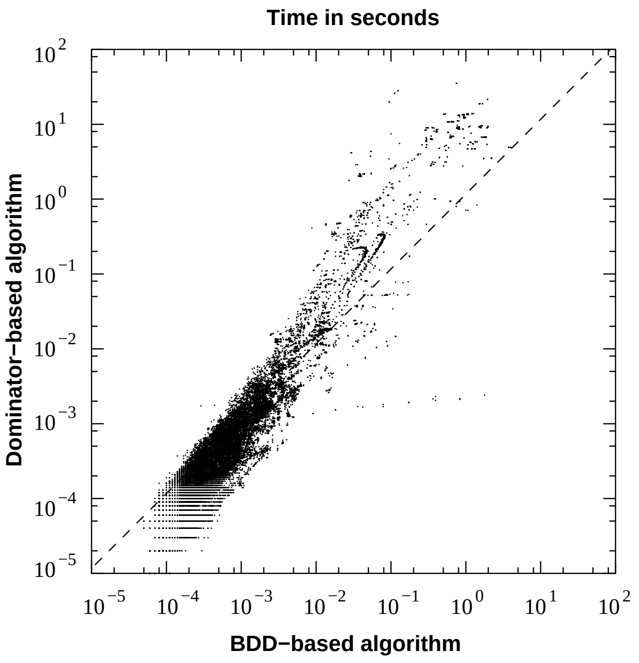

Figure 5: Runtime comparison.

8 Conclusions and Future Work

This paper presents an algorithm for evaluating the arithmetic transform of a Boolean function based on its circuit representation. Two distinct features make our algorithm feasible for large circuits. First, we use MTBDDs to represent intermediate results and to perform all arithmetic operations. Second, we use the dominator tree of the circuit to analyze and efficiently process re-convergent paths structures. The latter allows us to perform a stepwise evaluation of the arithmetic transform which keeps the size of the MTBDD at a minimum. This early reduction is not possible with previous approaches based on the construction of global BDDs.

Future work includes further improvements of the algorithms and exploring new applications in power analysis and functional verification. In particular we work on a more efficient algorithm for the COMPOSE operation by adapting the path-based approach used for BDD evaluation.

Acknowledgment

This work was supported in part by the Research Grant No 64700 from the SSF INTELECT program.

References

[1] R. Stankovic and T. Sasao, "A discussion on the history of research in arithmetic and reed-muller expressions," *Transactions on Computer-Aided Design*, vol. 20, pp. 446–455, September 2001.

[2] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *Transactions on Computers*, pp. 668–670, June 1975.

[3] F. Najm, "Transition density: A new measure of activity in digital circuits," *Transactions on Computer-Aided Design*, vol. 12, pp. 310–323, February 1993.

[4] J. Jain, J. Bitner, D. S. Fussell, and J. A. Abraham, "Probabilistic verification of Boolean functions," *Formal Methods in System Design*, vol. 1, pp. 63–117, 1992.

[5] S. Hassoun and T. Sasao, eds., *Logic Synthesis and Verification*. Boston, MA: Kluwer Academic Publishers, 2002.

[6] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, (Amsterdam, The Netherlands), pp. 193–207, March 1999.

[7] C. L. Berman and L. H. Trevillyan, "Functional comparison of logic designs for VLSI circuits," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 456–459, November 1989.

[8] T. Filkorn, *Symbolische Methoden für die Verifikation endlicher Zustandssysteme*. Dissertation, Technische Universität München, 1992.

[9] J. Moondanos, C. H. Seger, Z. Hanna, and D. Kaiss, "CLEVER: Divide and conquer combinational logic equivalence verification with false negative elimination," in *Computer Aided Verification (CAV'01)*, (Paris, France), pp. 131–143, July 2001.

[10] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," in *Proceedings of the 29th ACM/IEEE Design Automation Conference*, (Anaheim, CA), pp. 253–259, June 1992.

[11] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random pattern testability," *Transactions on Computers*, vol. C-33, no. 1, pp. 79–90, 1984.

[12] H. H. Aiken, "Synthesis of electronic computing and control circuits," *The annals of computational laboratory of Harward University*, vol. XXVII, 1951.

[13] Y. Komamiya, "Theory or relay networks for the transformation between the decimal and binary system," *Bull. Electrotech. Lab.*, vol. 15, no. 8, pp. 188–197, 1951.

[14] V. L. Artykhov, V. N. Kondratiev, and A. A. Shalyto, "Generating Boolean functions via arithmetic polynomials," *Automation and remote control*, vol. 49, pp. 508–515, April 1988.

[15] B. Falkowski and C.-H. Chang, "Generation of multi-polarity arithmetic transform from reduced representation of Boolean functions," in *1995 IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 2168–2171, 1995.

[16] B. Falkowski and C.-H. Chang, "Efficient algorithms for the calculation of arithmetic spectrum from OBDD and synthesis of OBDD from arithmetic spectrum for incompletely specified boolean functions," in *1995 IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 197–200, 1994.

[17] T. Sasao and e. M. Fujita, *Representations of discrete functions*. Kluwer Academic Publishers, 1996.

[18] T. Uchino, F. Minami, T. Mitsuhashi, and N. Goto, "On the complexity of using BDD's for the synthesis and analysis of Boolean circuits," in *Proceedings of 27th Annual Allerton Conference on Communication, Control and Computing*, pp. 730–739, September 1989.

[19] R. Ubar, J. Heinlaid, and L. Raun, "Improved testability calculation for digital circuits," in *Proceeding of NORCHIP*, pp. 264–270, 2001.

[20] M. G., "Bounding signal probabilities in combinational circuits," *Transactions on Computers*, pp. 1247–1251, October 1987.

[21] S. Ercolani, M. Favalli, M. Mamiani, P. Olivo, and B. Ricco, "Estimate of signal probability in combinational logic networks," in *Proceeding of IEEE European Test Conference*, pp. 132–138, 1989.

[22] R. Kapur and M. Mercer, "Bounding signal probabilities for testability measurement using conditional syndromes," *Transactions on Computers*, vol. C-36, pp. 1580–1588, December 1992.

[23] R. Kodavarti and D. Ross, "Signal probability calculations using partial functional manipulation," in *Proceedings of Eleventh Annual 1993 IEEE VLSI Test Symposium*, (Atlantic City, NJ, USA), pp. 194 – 200, April 1993.

[24] T. Uchino, F. Minami, M. Murakata, and T. Mitsuhashi, "Switching activity analysis for sequential circuits using Boolean approximation method," in *International Symposium on Low Power Electronics and Design*, (Monterey, CA), pp. 79–84, 1996.

[25] S. C. Seth, L. Pan, and V. D. Agrawal, "PREDICT-probabilistic estimation of digital circuit testability," in *Proceeding of International Symposium on Fault-Tolerant Computing*, pp. 220–225, June 1985.

[26] J. Costa, J. Monteiro, and S. Devadas, "Switching activity estimation using limited depth reconvergent path analysis," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 184 –189, 1997.

[27] J. Jain, M. Abadir, J. Bitner, D. Fussell, and J. Abraham, "Probabilistic design verification," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, (Santa Clara, California), pp. 468–471, November 1991.

[28] P. W. Purdom and E. F. Moore, "Immediate predominators in a directed graph," *Communications of the ACM*, vol. 15, pp. 777–778, August 1972.

[29] E. S. Lowry and C. W. Medlock, "Object code optimization," *Communications of the ACM*, vol. 12, pp. 13–22, January 1969.

[30] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translating, and Compiling, Vol. II.* Englewood Cliffs, NJ: Prentice-Hall, 1972.

[31] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *Transactions of Programming Languages and Systems*, vol. 1, pp. 121–141, July 1979.

[32] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. C.-Y. Yang, "Spectral transforms for large Boolean functions with application to technology mapping," in *Proceedings of the 30th ACM/IEEE Design Automation Conference*, (Dallas, TX), pp. 54–60, June 1993.

[33] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, (San Jose, CA), pp. 188–191, IEEE, November 1993.

[34] J. Monteiro, S. Devadas, A. Ghosh, K. Keutzer, and J. White, "Estimation of average switching activity in combinational logic circuits using symbolic simulation," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 121–127, January 1997.

[35] F. Somenzi, *CUDD: CU Decision Diagram Package, Release 2.3.1.* University of Colorado at Boulder, 2001.