Implementation of a Symbolic Circuit Simulator for Topological Network Analysis*

Weiwei Chen and Guoyong Shi
School of Microelectronics
Shanghai Jiao Tong University
Shanghai 200030, China
{chenweiwei, shiguoyong}@ic.sjtu.edu.cn

Abstract— **Many topological approaches to symbolic network analysis have been proposed in the literature, but none are implemented ultimately as a simulator for large network analysis due to their complexity and exponentially increasing number of terms. A novel methodology adopted in this paper uses a graph reduction approach based on a set of graph reduction rules developed recently. Furthermore, a Binary Decision Diagram is used in the implementation of a symbolic simulator that is capable of analyzing large analog circuit blocks. Implementation details and experimental results are reported.**

Keywords—**admissible term, BDD, graph reduction, symbolic analysis**

I. Introduction

Symbolic network analysis is a formal technique to calculate the behavior or the characteristics of a circuit in terms of symbolic parameters. In contrast to numerical simulators such as SPICE [11], symbolic simulators can provide insights to the circuit behavior, showing performance trade-offs and sensitivities to parameter variation and offering advantages in optimal topology selection, design space exploration and fault detection [1]. Symbolic network analysis from *topological* perspective has been studied extensively in the literature in the early 1950's [2], [3], [4] and recently [9]. One can find a comprehensive review on this approach in the textbook [5].

Despite the intensive research in the past decades on symbolic network analysis, most results did not come into practical simulators. The main difficulty arises from the exponential growth of the product terms with the number of nodes and elements in the circuit. A good symbolic simulator for *exact* analysis of large integrated circuit must have efficient ways to generate and store the product terms.

The simulator implemented in this paper is based on a graph reduction algorithm developed recently [7], together with an efficient storage scheme using Binary Decision Diagram (BDD) [6]. With a good symbol ordering heuristic, the graph reduction process can be represented by a BDD without exhausting the computer memory and it is no longer needed to explicitly enumerate the exponential number of terms. Furthermore, numerical analysis can be carried out efficiently due to the efficient implementation mechanism based on the BDD.

The work of this paper and [7] is based on the earlier work of [9] where a valid tree pair idea was proposed, but without a fully developed theory and a working implementation. This key contribution of this paper is on the implementation techniques based on an innovative graph reduction idea. The graph reduction algorithms are introduced in Section II. Implementation details on the symbolic simulator are presented in Section III. Experimental results are reported in Section IV. Conclusion is made in Section V.

*This work was supported by the National Natural Science Foundation of China, Grant No. 60572028.

II. Graph Reduction Algorithms

A graph reduction algorithm is presented with an example in this section. The circuits to be analyzed are allowed to contain elements such as impedances (Z), admittances (Y), four types of dependent sources (VCVS, CCCS, VCCS, CCVS), and independent source. To simplify formulation, the following assumptions are introduced:

Basic Assumptions

- A controlling branch only controls one branch.
- A controlled branch is controlled by only one branch.
- There is only one independent source in the circuit under analysis.

The assumptions are not restrictive; complex circuits can be remodeled to satisfy the assumptions in one way or another. For a given circuit satisfying the basic assumptions, it can be converted to a graph according to the following rules (see Fig. 1 for an illustration.)

Graph Construction Rules

(*i*) The edges associated with sources are directed, with voltage edges from + to - and current edges along the current direction assigned.
(*ii*) Add an edge for each controlling voltage. Each controlling current takes a single edge.
(*iii*) All edges are identified by their edge names.
(*iv*) The input-output is modeled by an appropriate dependent source.

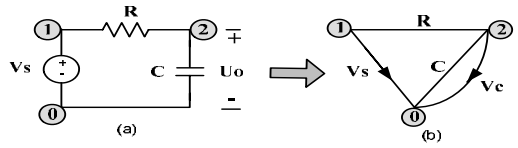

Fig. 1. A circuit example.

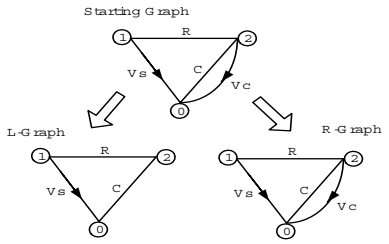

Fig. 2. Construction of Graph Reduction Diagram.

The graph construction rules basically assign a symbol to one independent edge or a pair of two dependent edges. In our graph reduction approach, the input-output pair is always modeled as a virtual dependent source, such as a VCVS (U_o controls V_s) for the example in Fig. 1(b), with the corresponding symbol solved symbolically as the unknown.

We continue to use the example in Fig. 1(a) to illustrate the construction of a decision diagram based on a graph reduction process. Because of the presence of dependent sources, splitting the graph into two subgraphs satisfying certain constraints would greatly simplify the topological analysis. Fig. 2 shows the splitting of the graph in Fig. 1(b) into two subgraphs, called L-graph and R-graph. According to definition of admissible term given in the Appendix, the V_c edge is only allowed in the R-graph.

The L-graph and R-graph are then reduced edge-by-edge following the *Graph Reduction Algorithm* until no further reduction is necessary. We choose an order for the symbols to be manipulated, X $<$ R $<$ C, where symbol X is associated with the VCVS pair and $X < R$ means manipulating symbol X before symbol R. Each symbol has two operations in the graph reduction process, one for adding this symbol into an admissible term (i.e., include the symbol) and the other for ignoring it (i.e., exclude the symbol). We begin with the symbol X which refers to a VCVS pair associated with the input-output. At the beginning, the initial L-graph and R-graph are intact (see Fig. 3).

The operations on the edges associated with different types of symbols are summarized in Table I, which are derived from the definition of admissible terms (see the Appendix).

TABLE I
BINARY OPERATIONS FOR GRAPH REDUCTION

	Include Symbol		Exclude Symbol	
	L-graph	*R-graph*	*L-graph*	*R-graph*
VCVS	Short VS	Short VC Open VS	Short VS	Short VS Open VC
CCVS	Short VS Open CC	Short CC Open VS	Short VS Short CC	Short VS Short CC
VCCS	Short CS	Short VC	Open CS	Open VC
CCCS	Short CS	Short CC	Short CC Open CS	Short CC
Y/Z	Short Y/Z	Short Y/Z	Open Y/Z	Open Y/Z

Since symbol X is associated with the VCVS pair, there are two types of operations listed in Table I for the edges V_s and V_c. Namely, one operation is to shorten the V_s edge and the V_c edge respectively in the L-graph and in the R-graph simultaneously as an edge-pair (meanwhile the V_s edge in the R-graph must be removed for consistency); the other operation is to shorten both V_s edges in the L-graph and the R-graph while removing the V_c edge from R-graph (i.e., in this case, the V_s edge is treated as a common edge). These two operations lead to respectively the left vertex R and the right vertex R pointed by the two signed edges rooted at the vertex X in the decision diagram (Fig. 3(a)). The resulting two reduced subgraph pairs are attached to the newly generated vertices (marked by "R"s) for further processing.

We continue to process the symbol R and then the symbol C by shortening and opening the relevant edges until no more symbols are left. Tracing along any path from the root to the terminal vertex in the decision diagram, if the number of shortened edges is equal to $N-1$, where N is the total number of nodes in the original graph, then the terminal vertex is marked "1"; otherwise, marked "0".

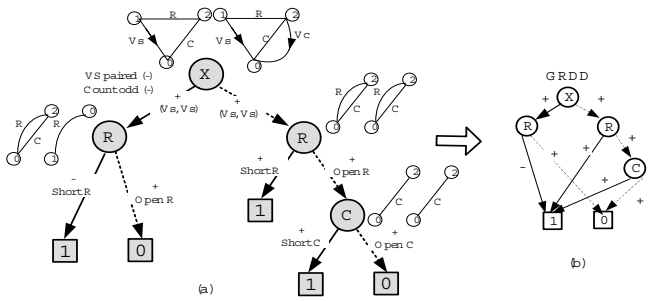

Fig. 3. Construction of Graph Reduction Decision Diagram.

The following *Graph Reduction Algorithm* summarizes the details for the graph reduction decision diagram construction. The *Sign Determination Algorithm* is used for the edge sign computation.

Graph Reduction Algorithm

Step 1. Initialization: Create *L-graph* and *R-graph* with VC edges removed from the L-graph and CS edges removed form the R-graph.

Step 2. Process the edges in the order specified for the symbols, with I/O being the first symbol. Short or open the edges according to the operations listed in Table I.

Step 3. Check the **Termination Condition**. If tree formed, point the BDD edge to the 1-terminal; If not, point it to the 0-terminal. Go to Step 6; otherwise, go to Step 4.

Step 4. Determine the signs attached to the BDD edges according to the **Sign Determination Algorithm**.

Step 5. Hash the subgraph-pair. If hashed, terminate the graph reduction along this BDD edge.

Step 6. More symbols unprocessed? If yes, go to Step 2; otherwise, quit.

Sign Determination Algorithm

Step 1. Initialization: Create two arrays, storing the L-graph and R-graph in that each edge is identified by its two end node numbers. Set $sign$:= 1.

Step 2. If the edge is opened, remove the edge from the arrays and keep the sign unchanged. If the edge (denoted $(v_1; v_2)$ with $v_1 < v_2$) is shortened, remove the edge from the arrays and replace the larger node number v_2 by the smaller node number v_1 of the shortened edge for all the remaining edges in the two arrays. Count the number of nodes in the two arrays that are indexed smaller than v_2. If the count is odd, update the sign by -1.

Step 3. If the edges being shortened are in *opposite direction* (denoted $(v_1; v_2)$ with $v_2 < v_1$), update the sign by -1.

Step 4. If the shortened edge is a VS and is not a common edge, update the sign by -1.

Step 5. Attach the resulting sign to the corresponding GRDD edge.

We note that each rooted path ending at the 1-terminal in the decision diagram represents an admissible term, which is the product of all the signed weights of the shortened edges. The graph reduction process results in a binary decision diagram consisting of symbol vertices and signed edges (see Fig. 3(b)), which is called a *Graph Reduction Decision Diagram* (GRDD). The BDD technique is also used in [8] but from a determinant-expansion point of view for symbolic circuit analysis.

In the graph reduction process, we shorten the edges by collapsing the larger end node into the smaller one meanwhile relabel the remaining edge nodes that have been collapsed in the subgraphs. The node renumbering is needed for the determination of GRDD edge signs, which is based on a recursive processing of the incidence matrices of a subgraph pair.

We refer the reader to [7] for a theoretical justification of the algorithms listed in this section. The correctness of the algorithms is also justified from implementation reported in the following sections.

III. Symbolic Simulator Implementation

Our symbolic simulator consists of a netlist parser, a symbolic analysis engine (containing the GRDD) and a numerical analyzer. The netlist parser reads the standard SPICE netlist and converts it to a directed graph stored in the computer memory according to the *Graph Construction Rules*.

The symbolic analysis engine processes the graph and constructs a binary decision diagram in the computer memory, in which the unknown symbol X is at the root vertex. For the circuit in Fig. 1, the symbols are ordered as $X > R > C$ and the GRDD constructed is shown in Fig. 3(b) with X being the virtual dependent pair (VCVS). The symbolic analysis engine generates the three product terms (*rooted paths* ending at the vertex 1) satisfying the homogenous equation (Theorem 1):

$$-X \cdot R^{-1} + R^{-1} + Cs = 0 \tag{1}$$

where X is the unknown. The transfer function from V_s to U_o is then $T(s) = 1/X$, where X is solved symbolically from (1) by sorting the terms.

In the GRDD, all terms involving the X symbol are stored as the *1-edge* sub-diagram and those terms not involving the X symbol are stored as the *0-edge* sub-diagram of the GRDD root. The analyzer evaluates the product terms stored in the GRDD with all symbols substituted by their (complex) numerical values (with the Z values inverted) and divides the value of 1-edge by that of the 0-edge at the root to get the one-point frequency response. The analyzer also calculates the statistics of the decision diagram, including the number of vertices created and the number of terms it represents, etc.

The GRDD construction details have been articulated in the previous section. Described below are the implementation details that are critical for GRDD efficiency.

A. Symbol Ordering Heuristic

The symbol processing order strongly affects the size of GRDD. In the current implementation, the symbols of the circuit elements are ordered starting from the I/O port, then the elements directly connected to it, and then the elements connect to the previously ordered elements until all of the symbols are ordered. This process of ordering is easily implemented in a breadth first fashion. We define some priorities for those equivalent elements (elements that are all connected to the previously ordered ones). The impedances (admittances) are assigned the higher priority but ordered at random. This ordering heuristic is from the consideration of early termination, namely, graph disconnectivity can be detected early. We believe other better good orderings exist by exploring the circuit topology.

B. GRDD Sharing

As usual, the efficiency of BDD implementation comes from the sub-diagram sharing. In the same vein, the GRDD sharing is considered in our implementation as well. The sub-GRDD sharing comes from the fact that in the graph-pair reduction process, some later reduced graph-pairs will find themselves identical or topologically isomorphic to the earlier reduced graph-pairs.

In GRDD construction, we always merge two end nodes of an edge by retaining the smaller node number. This convention would end up with certain reduced subgraphs (from different reduction paths) having the same topology but different node numberings. We call such reduced subgraphs *isomorphic subgraphs* (see Fig. 4(a) for an example). Furthermore, as shown in Fig. 5, although the two subgraph pairs have different topologies, they lead to two identical sub-GRDDs. We call such reduced subgraphs *term-equivalent* subgraphs.

It is easy to observe that both isomorphic and term-equivalent subgraphs would result in the same set of admissible sub-terms (see Fig. 4(b) and Fig. 5 for examples), regardless of the node numbering and even the subgraph topologies.

Subgraph isomorphism and term-equivalence could possibly lead to sub-GRDD sharing. In our implementation, identical subgraph pairs are shared first, followed by considering whether the associated GRDD vertices can be shared. GRDD vertices are shared when their attached reduced subgraph pairs, symbol indexes and the signs attached to their incident edges are all identical to each other. The two sharings are implemented by hash functions. The hash key for the identical subgraph sharing is determined by the topology of the subgraph pair, while the hash key for the vertex sharing is determined by the attached reduced subgraph pair, symbol index and the sign of the incident edge. Hash tables are used to keep each reduced subgraph-pair and GRDD vertex unique.

Note that the vertex sharing implemented above did not consider the possible sharing resulting from isomorphic and term-equivalence. For better efficiency, this part of vertex sharing is implemented in the second phase called *GRDD reduction* explained in the next subsection.

C. GRDD Reduction

GRDD Reduction takes care of possible vertex sharing resulting from subgraph isomorphism or term-equivalence. In our implementation, we used a vertex labeling technique. The label of a GRDD vertex is determined by the labels of its children vertices. We set the 0-terminal with label 0 and 1-terminal with label 1. *GRDD Reduction* keeps the vertex labels unique in GRDD and *unreference* the redundant ones. *GRDD Reduction* also reduces any GRDD vertex whose 1-edge and 0-edge both point to the 0-terminal; it simply replaces such a vertex by the 0-terminal and unreferences it. After reduction, we do a *garbage collection* to free those unreferenced vertices. Fig. 4 shows an example for *GRDD Reduction* on isomorphic sub-diagram.

Reduction and Garbage Collection are standard manipulations in BDD packages [6].

D. GRDD Evaluation

Numerical frequency response can be obtained easily from the GRDD constructed by substituting the symbols with their numerical values at a set of frequency points. The efficiency of this symbolic simulator is determined by two parts: the GRDD construction and the numerical evaluation. The efficiency of GRDD construction largely depends on a good symbol ordering scheme. As long as a GRDD can constructed in reasonable time, the time for numerical evaluation is negligible due to the efficiency by the hashing mechanism implemented.

The partial numerical value at an GRDD vertex is evaluated recursively by the following formula:

$$\begin{aligned} eval(vertex) = {} & eval(vertex{\rightarrow}left) * V(symbol) * sign_1(vertex) \\ & + eval(vertex{\rightarrow}right) * sign_0(vertex) \end{aligned} \tag{2}$$

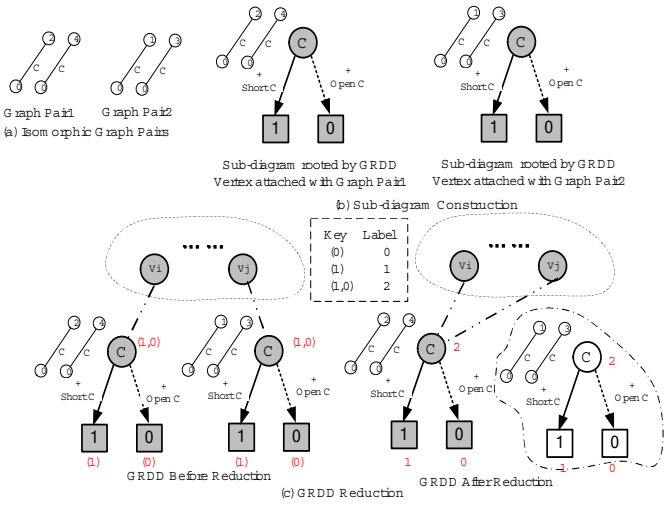

Fig. 4. An example for GRDD reduction.

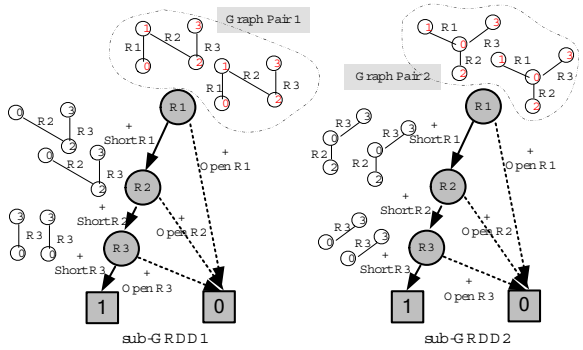

Fig. 5. Identical sub-GRDDs generated by topologically different graph pairs.

where $vertex$ is any GRDD vertex, $eval(vertex)$ is the complex value calculated at vertex $vertex$, $vertex{\rightarrow}left$ points to the child vertex connected by the 1-edge and $vertex{\rightarrow}right$ points to the child vertex connected by the 0-edge, $sign_1$ and $sign_0$ are respectively the signs attached to the 1-edge and 0-edge, and $V(symbol)$ is the numerical value of the symbol at the GRDD vertex (with the Z value inverted). With hashing techniques, the complexity of recursive numerical evaluation is linear in the GRDD size.

E. Strategies for Efficiency

1) Lumping parallel branches: The circuit to be analyzed usually contain parallel elements that appear as parallel edges in the converted graph. Lumping these parallel branches to one branch means multiple symbols are combined into one symbol, by which the graph complexity and the number of product terms are reduced remarkably.

2) Early Disconnectivity Detection: Part of the *Termination Condition* in the *GRDD Construction Algorithm* is by detecting the graph disconnectivity as early as possible. We count the number of the edges in the reduced graph by counting those parallel edges as one edge and ignoring those self-looped edges. If the number of edges is less than the number of the vertices of the reduced graph *minus* one, then the reduced graph is disconnected and the current GRDD vertex should be pointed to the 0-terminal.

IV. Experimental Results

Our simulator was implemented in C++ and run on Intel Pentium 1.73GHz processor with 1G memory. Three benchmark circuits used in our experiment are:

- **μa741**, a bipolar opamp containing 24 transistors (same circuit used in [8], Fig. 15.)
- **μa725**, a bipolar opamp containing 26 transistors (same circuit used in [10], Fig. 13.)
- **MOSopamp**, a MOS cascode opamp containing 22 transistors (same circuit used in [10], Fig. 8.)

The small signal models for the MOSFET and bipolar transistors are the same as that used in [8], Fig. 14, or in [10], Fig. 4.

Table II shows the simulator performance of the sample circuits. $edge\_lump$ is the number of edges after the parallel elements are lumped. $|GRDD|$ is the size of GRDD. Note that for the μa725 circuit we used a different ordering heuristic than the one described before. A universally applicable ordering heuristic is still under investigation.

Table II also shows the effects of *GRDD Sharing* and *GRDD Reduction*. The number of GRDD vertices is much less after GRDD reduction comparing to that before GRDD reduction. *GRDD Sharing* happens frequently during the construction.

Shown in Fig. 6 is a set of frequency responses produced by our symbolic simulator and HSPICE. The results of our simulator match exactly those obtained from HSPICE [12]. These results provides part of justification for the correctness of our symbolic simulator.

The experimental results show that our symbolic simulator is capable of generating the exact network functions of large analog circuits, such as μa741 and μa725, in the scale of seconds. (Note that in [10] the authors use approximate techniques to analyze μa725.)

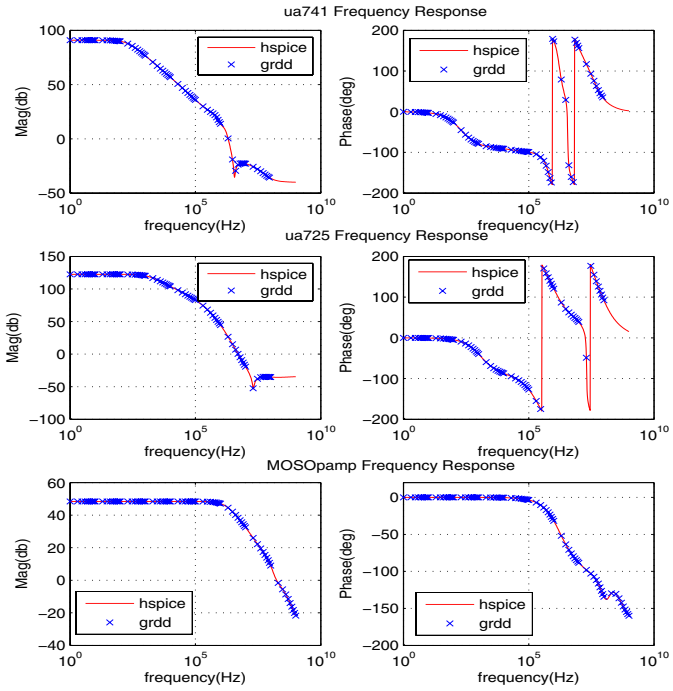

Fig. 6. Frequency responses of the benchmark circuits (hspice vs. grdd).

TABLE II
SIMULATOR PERFORMANCE FOR BENCHMARK CIRCUITS

Ckt name	#Edge	#Edge_lump	#Node	#Symb	#Term	$\|GRDD\|$	Time	Memory
μa741	160	103	24	81	1.39e+14	31887	1.9s	34.78MB
μa725	166	120	31	98	5.09e+17	53420	22.6s	358.7MB
MOSopamp	182	90	14	65	2.93e+09	154452	4.3s	84.52MB

Ckt name	#Reduced graph sharing	#GRDD vertices sharing	#GRDD vertices without reduction	#GRDD vertices with reduction
μa741	209005	35870	85815	31887
μa725	4928831	858184	1753994	53420
MOSopamp	81652	147337	334423	154452

Shi and Tan [8] also reported that their DDD-based symbolic simulator can analyze the μa741 circuit exactly. The DDD package uses determinant-based formulation, lacking the topological information during symbolic analysis, whereas our simulator directly manipulates on the circuit topology, thus can potentially provide the designer more intuition on the circuit under design. With good symbol ordering, both simulators are able to construct decision diagrams in less than a minute.

V. CONCLUSION

This paper has described some implementation details on an efficient symbolic simulator based on a graph reduction approach. The graph reduction process is designed in such a way that using a BDD can avoid processing the exponential number of symbolic product terms explicitly, meanwhile numerical evaluation also can be carried out efficiently due to the sharing mechanism offered by BDD. This simulator is the first one ever capable of analyzing large analog circuits directly from the circuit topology. Finally, we point out that this symbolic simulator can be extended easily to deal with ideal opamps modeled by nullators and norators.

APPENDIX

A. Basic Concepts and Main Theorem

This appendix provides an outline of the theoretical foundation of this work. The proof of the main theorem is available in [7].

Definition 1 (Admissible Tree-Pair) *An admissible tree-pair consists of an L-tree and an R-tree with the following conditions satisfied:*

(*i*) *All Y and Z edges appearing in an admissible tree-pair are common edges.*

(*ii*) *All CC and VS edges in the original network must appear in the admissible tree-pair, but are allowed to appear either as common edges or as pairing edges,* exclusively. *If appearing in pair, CC edges must be in the R-tree while VS edges must be in the L-tree.*

(*iii*) *Any VC and CS edges may or may not appear in the admissible trees. However, whenever they appear, they must appear as pairing edges with the VC-edge in the R-tree and the CS-edge in the L-tree.*

When all edges are identical in the two trees of an admissible tree-pair, the admissible tree-pair reduces to an *admissible tree*. Hence, admissible tree is a special case of admissible tree-pair.

Definition 2 (Admissible Term) *The signed product of all edge weights from an admissible tree-pair or an admissible tree is called an admissible term. The edge weights are defined as follows:*

(*i*) *Common edges: The weight for a Y edge is the element symbol Y. The weight for a Z edge is the reciprocal of the element symbol, Z^{-1}, and the weights for all common CC and VS edges are* one.

(*ii*) *Pairing edges: The weights for four dependent sources are signed multipliers defined as: $-E$ for VCVS, $+F$ for CCCS, $+G$ for VCCS, and $-H$ for CCVS.*

(*iii*) *Term sign: The term sign for an admissible tree is always positive. The term sign for an admissible tree-pair is determined by the* ***Sign Determination Algorithm****.*

Theorem 1 (Main Theorem) *Under the* Basic Assumptions*, all admissible terms defined by Definition 2 sum up to zero and are cancellation-free.*

REFERENCES

[1] G. Gielen, P. Wambacq, and W. Sansen, "Symbolic analysis methods and applications for analog circuits: A tutorial overview," *Proceedings of the IEEE*, vol. 82, no. 2, pp. 287–303, Feburary 1994.

[2] S. Mason, "Feedback theory – further properteis of signal flow graphs," *Proc. IRE*, vol. 44, pp. 920–926, July 1956.

[3] W. Mayeda and S. Seshu, "Topological formulas for network functions," University of Illinois, Urbana, Tech. Rep. Engineering Experimentation Station Bullitin 446, 1959.

[4] A. Talbot, "Topological analysis of general linear networks," *IEEE Trans. on Circuit Theory*, vol. CT-12, no. 2, pp. 170–180, June 1965.

[5] P. Lin, *Symbolic Network Analysis*. New York: Elsevier, 1991.

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

[7] G. Shi, W. Chen, and C.-J. Shi, "A graph reduction approach to symbolic circuit analysis," submitted to *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2006.

[8] C.-J. Shi and X.-D. Tan, "Canonical symbolic analysis of large analog circuits with determinant decision diagrams," *IEEE Trans. on Computer-Aided Design*, vol. 19, no. 1, pp. 1–18, January 2000.

[9] Z. Yin, "Symbolic network analysis with the valid trees and the valid tree-pairs," in *IEEE Int'l Symposium on Circuit and Systems*, Sydney, Australia, 2001, pp. 335–338.

[10] Q. Yu and C. Sechen, "A unified approach to the approximate symbolic analysis of large analog integrated circuits," *IEEE Trans. on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 43, no. 8, pp. 656–669, 1996.

[11] L.W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits", Ph.D. dissertation, Univ. California, Berkeley, CA, May 1975.

[12] Synopsys, Inc. "HSPICE - The golden standard for Accurate Circuit Simulation" *http://www.synopsys.com/products/mixedsignal/hspice/hspice.html*