A Graph Reduction Approach to Symbolic Circuit Analysis

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

C.-J. Richard Shi
Department of Electrical Engineering
University of Washington
Seattle, WA 98195, U.S.A.
cjshi@ee.washington.edu

***Abstract*— A new graph reduction approach to symbolic circuit analysis is developed in this paper. A Binary Decision Diagram (BDD) mechanism is formulated, together with a specially designed graph reduction process and a recursive sign determination algorithm. A symbolic analog circuit simulator is developed using a combination of these techniques. The simulator is able to analyze large analog circuits in the frequency domain. Experimental results are reported.**

I. Introduction

Symbolic circuit analysis is concerned with finding a network function (transfer function) from one signal to another expressed in terms of the circuit element symbols. While many methods have been studied starting from 1950's, such as [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] among others, few of them are actually in use today. As pointed out in [11], most of the classical methods are unable to deal with large-scale integrated circuits due to the unscalable complexity in implementation. For a review on classical and contemporary methods, refer to the monographs [12], [13], [14] and the survey paper [15].

Recent research efforts on symbolic methods target at large network blocks containing 20 to 40 transistors, typical for most analog blocks. Analog designers tend to use approximately derived formulas in design practice, but would like to justify their approximate analysis by comparing with the analytical formulas derived from a symbolic simulator. Except for numerical results, Spice-like simulators are not able to provide analytical formulas such as gain and sensitivity expressions. However, deriving analytical expressions for a network consisting of about 20 transistors (with typical small-signal models) is a daunting task for all symbolic simulators [16], [17].

Reduced Ordered Binary Decision Diagram (ROBDD) proposed by Randal Bryant [18] opened a new door to many NP-hard problems in the design automation community. The idea of determinant decision diagram (DDD) proposed by Shi and Tan [19] was the first successful application of decision diagram to symbolic manipulation of the explosive number of cofactors in the expansion of a network determinant. However, since DDD is MNA based, its efficiency is limited by the duplicated symbols and cancellation. Hence the size of circuits DDD can analyze without using approximation is limited.

Another alternative approach is to construct a decision diagram directly from the circuit topology, which is the subject of this paper. By introducing a successive graph reduction and sign determination scheme, one can construct a decision diagram efficiently by sharing sub-networks.

This paper starts from introducing the basic idea and steps for graph reduction using a simple RC example in section II. The general graph reduction process is formulated in section III. Algorithms for decision diagram construction and sign determination are presented in section IV, followed by simulator construction and simulation results in section V. This paper concludes in section VI. A theoretical foundation of the graph reduction method is presented in the appendix.

*The work of G. Shi and W. Chen was supported by the National Natural Science Foundation of China, Grant No. 60572028.

II. Preliminary

Two main approaches for systematic computerized symbolic analysis are algebraic methods and topological methods. Signal flow graph is a topological approach, but the SFG constructed may have less resemblance to the original circuit. In fact, there is another topological approach that solves a network function directly from the given circuit topology, at most with slight modifications. This approach would be beneficial to analog designers who usually are used to manipulation of circuit topology. The symbolic analysis technique to be developed in this paper follows the latter approach.

A. Review of the tree-pair idea

To help understand the basic idea of the current approach, we start from the simple example shown on the left side in Fig. 1. For this circuit, the transfer function from the voltage source V_s to the voltage across the capacitor V_c is $H(s) = 1/(1 + RCs)$. The reduction approach proceeds as follows. First draw a graph resembling the circuit (right side in Fig. 1), where an extra edge V_c is added across the capacitor. This extra edge V_c and the edge V_s form a *virtual* voltage controlled voltage source (VCVS) pair, denoted symbolically as $V_s = XV_c$. The transfer function is solved if the unknown X is solved symbolically, i.e., $H(s) = V_c/V_s = 1/X$.

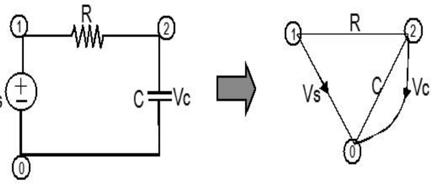


Fig. 1. An RC circuit and its graph (Example 1).

An idea of valid tree and tree-pair was proposed recently by Yin *et al* [20], [21]. By *valid* it means that only those trees and tree-pairs satisfying certain constraints are enumerated. In this sense it is more appropriate to call such trees and tree-pairs *admissible*.

Admissible trees or tree-pairs are constructed according to certain rules, which were stated in the papers [20], [21]. Deriving such rules is a nontrivial task. Unfortunately, no derivation is available in [20], [21] and rules were stated in very vague language. Nevertheless, the basic idea there turns out to be valuable in that it can be improved and reformulated to make the method practically applicable. In this paper we derive the constraints for all admissible trees and tree-pairs from a rigorous algebraic treatment (see the appendix), based on which some efficient algorithms are designed for implementation. Before we get into the formality, we continue to articulate the basic techniques using the working example.

For the preceding example, two admissible trees and one admissible tree-pair are found in Fig. 2. Three signed terms are then derived from the trees and tree-pair as R^{-1}, Cs, and $-XR^{-1}$, respectively, according to certain rules. By summing them up to zero, one solves that $X = 1 + RCs$, by which the transfer function is $H(s) = 1/(1 + RCs)$.

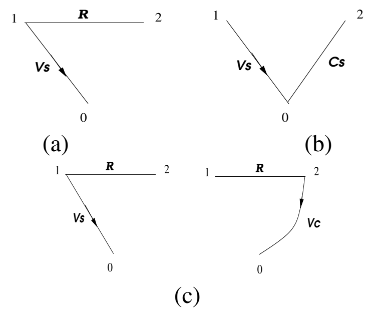


Fig. 2. (a,b) Admissible trees. (c) Admissible tree-pair.

The valid tree and tree-pair idea proposed by Yin *et al* is not directly applicable to large circuits, because this method is enumeration based, suffering from the exponentially increasing number of product terms.

B. Improvement on the tree-pair enumeration

To get around the curse-of-dimensionality, we shall introduce a creative idea by converting the tree-enumeration process into a graph reduction process, meanwhile the reduction process is implemented by a decision diagram (BDD) data structure which takes the advantage of sub-graph sharing.

The graph reduction process is illustrated once again by our working example. An admissible tree-pair consists of a left-tree (L-tree) and a right-tree (R-tree). Since the graph edges allowed to be tree edges of an admissible tree-pair are not all identical (see Fig. 2), it is convenient to crate at the beginning two subgraphs from the original graph, with the left-graph (L-graph) containing all edges allowed for all L-trees, while the right-graph (R-graph) containing all edges allowed for all R-trees. For the current example, the L-graph and R-graph constructed are shown in Fig. 3, where according to the definition of admissible tree-pairs (see appendix), V_c is allowed on the R-graph only.

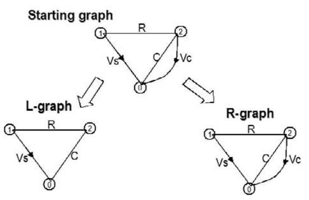


Fig. 3. Left and right subgraphs.

The L-graph and R-graph are then reduced successively by following a set of graph reduction operations until no further reduction is necessary. We leave the details of the reduction operations to the next section, but just to illustrate the basics here. The graph reduction process shown in Fig. 4 is explain as follows.

First we choose an order for the symbols, $X < R < C$, with $A < B$ meaning A precedes B. Attached to vertex X is the pair of beginning L-graph and R-graph. Since X represents the dependent VCVS pair (V_s, V_c), there are two operations available for it, one by shortening V_s edge in L-graph (meanwhile removing V_s edge from R-graph) and shortening V_c edge in R-graph (meanwhile removing edge C from R-graph); the other by removing V_c edge from R-graph first then shortening both V_s edges from L-graph and R-graph. These two operations lead to the left vertex R and the right vertex R in the decision diagram, respectively, with the two arrows signed. The resulting reduced graphs are attached to the left and right vertex R, respectively. Further operate on the reduced graphs by shortening/opening the edge R and then the edge C, until no more edges available. By tracing the decision diagram along a path from the root to the leaf, if the number of shortened edges is equal to $N-1$, where N is the total number of nodes in the original graph, the leaf node is marked "1", otherwise it is marked "0". We note that the path ending at "1" represents an admissible tree or an admissible tree-pair, and the product of all signed weights (for the shortened edges) along the path is the admissible term needed to form the sum-of-product expression.

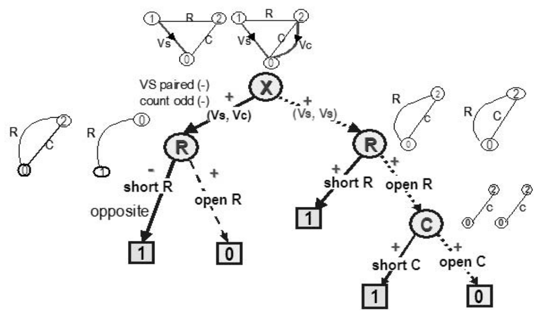


Fig. 4. Illustration of graph reduction.

III. Formulation of Graph Reduction

The graph construction and reduction steps outlined above is formalized now. We find the following assumption sufficient for practical use.

Assumption 1 (Basic Assumption)

- *The circuit is linear and can be modeled by elements of impedances, admittances, the four types of dependent sources, independent source, and nullators and norators.*
- *A dependent source pair involves only one controlling branch and one controlled branch, and no self-controlling and mutual-controlling exists.*

A graph is created from the circuit network according to the following rules.

Graph Construction Rules

(*i*) Controlling or controlled edges are directed; *a voltage edge is directed from + to −, and a current edge is directed along the assigned current direction.*

(*ii*) A controlling voltage is identified by an extra edge in parallel to the circuit element that has that voltage. (Note that no current flows through such an added edge.)

(*iii*) A controlling current edge is identified by an extra edge in series with the circuit element, through which the current flows. (Note that no voltage drops across such an added edge.)

(*iv*) An ideal opamp is modeled by a nullor pair, consisting of a nullator and a norator and represented respectively by an *NU* (nullator) edge and an *NO* (norator) edge in the graph.

Throughout the paper, we shall be using shorthand such as CC (current controlling), VC (voltage controlling), CS (current source), and VS (voltage source) to refer to the edges involved in the VCVS, CCCS, VCCS, and CCVS pairs, and NU and NO to refer to the edges of nullors.

The definition of admissible tree-pair (Definition 1 in the Appendix) requires that certain edges be allowed only in one of the trees in a tree-pair while other edges be allowed in both R-tree and L-tree. To facilitate graph reduction, we adopt a graph splitting strategy, i.e., to split the original graph into two subgraphs called *L-graph* and *R-graph*, where each subgraph contains all the edges allowed for it. (see Fig. 3). Listed in Table I are the detailed operations for different types of the graph edges. The two columns labeled *SHORT* and *OPEN* indicate the two generic types of decision to be taken at each vertex of the decision diagram.

We remind that, since CC and VS edges are allowed in both L-graph and R-graph, whenever CC or VS is shortened in one graph as a pairing edge, it has to be removed from the other graph for consistency.

TABLE I
OPERATIONS FOR GRAPH REDUCTION.

	SHORT	OPEN
Y or Z	Short Y/Z	Open Y/Z
E (VCVS)	Short VC; Short VS	Open VC; **Short VS**
F (CCCS)	Short CC; Short CS	**Short CC**; Open CS
G (VCCS)	Short VC; Short CS	Open CC; Open CS
H (CCVS)	Short CC; Short VS	**Short CC**; **Short VS**
(NU,NO)	Short NU; Short NO	Open NU; Open NO

The edge operations listed in Table I are derived directly from the definition of admissible tree-pairs and Theorem 1 in the appendix. Suppose the graph nodes are numbered continuously from $0, 1, 2, \cdots, N-1$ with the node 0 being the ground. We specify that, when shortening an edge, the node resulting from collapsing two nodes is relabeled by *retaining the smaller node number*.

IV. Graph Reduction Algorithms

A. Graph Reduction Decision Diagram

A graph pair is reduced successively following a prespecified symbol order. The reduction process is essentially a binary decision process, at each decision point two generic operations *SHORT* and *OPEN* are made, each leading to a sub-diagram. To avoid the exponential growth of the diagram structure, binary decision diagram (BDD) manages to take the advantage of subgraph sharing. In fact, in the reduction process different reduction paths may end up with *isomorphic* subgraph pairs in the sense of identical topology but possibly different node labeling. A typical technique used in implementation is hashing [22].

The decision diagram created in the process of graph reduction is called a *Graph Reduction Decision Diagram* (GRDD). In a GRDD, each diagram arrow is associated with a reduction operation together with a sign, meanwhile each diagram vertex is associated with a symbol. The two arrows originating from a vertex correspond to the two operations for the symbol in the vertex.

Standard BDD terminologies are used in the algorithm description below. The solid arrow in a GRDD is called the *1-edge*, corresponding to the operations listed under the column *SHORT* in Table I, while the dashed arrow is called the *0-edge*, corresponding to the operations listed under the column *OPEN* in Table I. A path from the root to the *1-terminal* is called a *1-path*. An admissible term is formed by all the symbols issuing a solid arrow (i.e. 1-edge) along a 1-path. The sign of an admissible term is the product of all the signs along the 1-path.

At the end of graph reduction, the decision diagram obtained is purely a BDD containing all circuit symbols and the necessary sign information. (see Fig. 4.) Such a decision diagram is also called a *Symbol Decision Diagram* (SDD), in which information on the intermediate reduced graph pairs is immaterial. The SDD size is determined by the number of vertices in the SDD (excluding the terminal vertices "1" and "0").

The following GRDD construction algorithm summarizes all the graph reduction details described so far. To facilitate symbolic analysis, we always order the unknown symbol X first.

GRDD Construction Algorithm:

```
Step 1 Initialization: Create the L-graph and R-graph, each
       containing all edges allowed.
Step 2 Process the edges in the order specified for the sym-
       bols with the symbol for I/O goes first. Short or open
       the appropriate edges associated with the symbol ac-
       cording to the operations listed in Table I.
Step 3 Check the Termination Condition: If tree formed,
       point the edge to the terminal node "1"; then go to Step
       6. If no tree possible, terminate the edge at the node
       "0"; then go to Step 6. Otherwise, go to Step 4.
Step 4 Determine the signs for the SHORT-edge (1-edge) and
       the OPEN-edge (0-edge), respectively, according to
       the Sign Determination Algorithm (see next).
Step 5 Check subgraph isomorphism and hash the subgraph
       pair pointed to by the current decision edge. If hashed,
       connect the pointer and terminate the graph reduction
       process following this decision edge.
Step 6 More symbols unprocessed?
       If yes, goto Step 2. Otherwise, quit the algorithm.
```

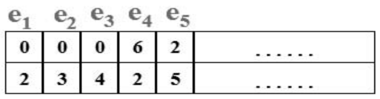
B. Sign determination

Sign determination is a crucial part in the graph reduction process. Let A_L and A_R be the two reduced incidence matrices of admissible L-tree and R-tree, respectively, with their rows (labeled by node numbers) in exactly the same order and their columns (labeled by the edge names) aligned. The following proposition is well-known in the literature.

Proposition 1 *The sign for an admissible term determined by an admissible tree-pair is the product of the determinants of A_L and A_R, i.e.,* $\det|A_L| \cdot \det|A_R|$. *(Note that each determinant evaluates to either $+1$ or -1.)*

Next we introduce a recursive sign determination procedure that can be integrated with the graph reduction and sharing process.

In implementation, two 2-dimensional arrays are used to represent the edges in the L-graph and R-graph, respectively, where each edge is identified by its two end-node numbers (Fig. 5). Let $e(1)$ and $e(2)$ be the first (upper entry) and the second nodes (lower entry) of edge e, respectively. Assume all edges are directed from the node $e(1)$ toward $e(2)$, including those Y or Z edges. Because of the edge-merging operation and node relabeling, an edge identical in the two subgraphs could have distinct end-node numbers but it does not affect sign determination. In an edge-shortening operation, we always merge the *larger* node number toward the *smaller* node number, while keeping the smaller node number. The two edges (either common or paired) are said in *opposite direction* if in one column $e(1) < e(2)$, while in the counterpart column $e(1) > e(2)$. The sign determination algorithm is summarized below.



e_1	e_2	e_3	e_4	e_5	
0	0	0	6	2	......
2	3	4	2	5	......

Fig. 5. A subgraph represented in a 2D-array.

Sign Determination Algorithm

Step 0 Initialize: set $sign := 1$.

Step 1 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 for all the remaining edges in the two arrays that are connected to v_2. Count the number of nodes in each array (excluding v_1 and v_2) that are indexed smaller than v_2. If the count is odd, update the sign by -1. (Count the two arrays separately.)

Step 2 If the two edges being shortened are in *opposite direction*, update the sign by -1.

Step 3 If the shortened edge is a VS and is not a common edge, update the sign by -1.

Step 4 Attach the resulting sign to the corresponding SDD edge.

The sign determination algorithm is a direct consequence of the Gaussian elimination process of an incidence matrix. Its proof is omitted.

V. Simulator Construction

The Symbol Decision Diagram (SDD) constructed using the algorithms presented serves as the symbolic analysis engine for our symbolic simulator. The underlying principle is explained here. By the main theorem in appendix (Theorem 1), the summation of all signed admissible terms is equal to zero. For the designated input-output of a circuit, the network transfer function is identified by a virtual controlled source with the gain symbol X. The set of all admissible terms can be divided into two subsets, one containing the terms with symbol X while the other containing the rest. Algebraically, we have the following expression

$$X\left(\sum_{i=1}^{m_1} t_i\right) + \left(\sum_{j=1}^{m_2} T_j\right) = 0. \quad (1)$$

where t_i are those signed terms with X removed and T_j are the rest of the signed terms without X. Then X can be solved as

$$X = -\frac{\sum_{j=1}^{m_2} T_j}{\sum_{i=1}^{m_1} t_i}. \quad (2)$$

The decision diagram constructed, with the unknown symbol X as the root vertex, is a symbolic representation of the network function, from which other symbolic analyses can be done, such as plotting frequency response curves, plotting poles-zeros, deriving approximate gain functions and sensitivity function, etc. Note that in the SDD, the partial sum $\sum_{i=1}^{m_1} t_i$ is stored at the *1-edge* of the vertex X, while the partial sum $\sum_{j=1}^{m_2} T_j$ is stored at the *0-edge* of the vertex X. The numerical value of each product term is the product of all symbols substituted by their (complex) numerical values (with the Z values inverted). At each specified frequency point, the two sub-SDDs pointed to by vertex X are evaluated separately and then the value for X is obtained from formula (2). The efficiency of numerical evaluation is improved greatly by implementing hashing and cache mechanism (see [23] for more details on implementation.)

Our symbolic simulator was implemented in C++ and run on Intel Pentium 1.73GHz processor with 1G memory. A Spice-compatible netlist grammar is used. The simulator starts from parsing a netlist, creates an internal graph, then initiates a graph reduction process until a decision diagram is created.

The performance of a symbolic simulator is measured by how much time and memory it takes to build an internal representation of the symbolic transfer function and how fast it is in the numerical evaluation phase.

Our symbolic simulator has been used to analyze two typical analog blocks, amplifier $\mu A741$ (24 transistors, Fig. 6) and $\mu A725$ (26 transistors, schematic omitted). The user is allowed to choose (define) small signal models. For example, the one shown in Fig. 7 was used in our simulation.

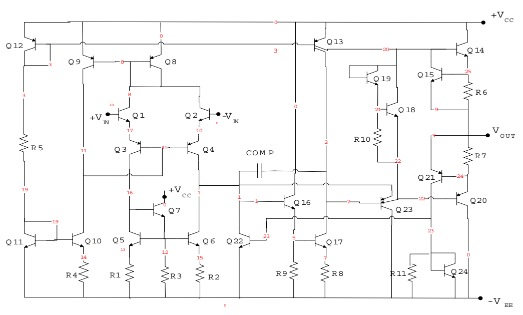


Fig. 6. Schematic of $\mu A741$.

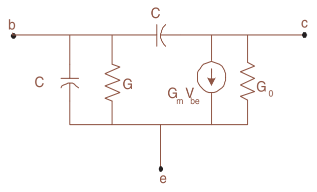


Fig. 7. BJT small signal model.

Table II shows the simulator performance collected for the $\mu A741$ circuit, where E is the number of edges in the graph, E_{lump} is the number of edges after the parallel edges are lumped, N is the number of nodes in the graph, $\#term$ is the number of product terms, $\#symb$ is the number of symbols in the circuit, $|SDD|$ denotes the size of SDD. The simulator performance for the $\mu A725$ circuit is shown in Table III. We note that these performance results were collected under an appropriate ordering of the symbols in both circuits. The ordering heuristics is also a subject of research, the more details on which are to be reported elsewhere.

TABLE II
SIMULATOR PERFORMANCE FOR CIRCUIT $\mu A741$.

E	E_{lump}	N	#term	#symb	$\|SDD\|$	Time	Memory
160	103	24	1.39e+14	81	57687	3.2s	59.6MB

TABLE III
SIMULATOR PERFORMANCE FOR CIRCUIT $\mu A725$.

E	E_{lump}	N	#term	#symb	$\|SDD\|$	Time	Memory
166	120	31	5.09e+17	98	53420	22.6s	358.7MB

VI. Concluding Remarks

We have developed an efficient symbolic analysis technique that implements a graph reduction reduction process by using the decision diagram mechanism. This technique has been used to construct an efficient symbolic analysis engine for analyzing a linearized analog integrated circuit in the frequency domain. Experimental results have demonstrated the efficiency of the proposed theory and implementation. Future work includes efficient implementation on symbolic poles/zeros analysis, sensitivity function extraction, and approximate gain analysis, etc.

Appendix

A. Main Theorem

First admissible tree-pairs and trees are defined, followed by the definition of admissible terms. The main theorem of this paper is then stated with the main proof given.

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 NU and NO edges in the original network* must *appear on all admissible tree-pairs, with the NU edges in the R-tree and the NO edges on the L-tree.*

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

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

(*iv*) *Any VC and CS edges may or may not appear on 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: For Y edges the weights are Y's, for Z edges the weights are Z^{-1}, and for all common CC and VS edges the weights are* one.

(*ii*) *Pairing edges: The following signed multipliers are used for the weights of dependent pairs: $-E_{j,k}$ for VCVS, $+F_{j,k}$ for CCCS, $+G_{j,k}$ for VCCS, and $-H_{j,k}$ for CCVS. Each nullor pair is weighted one as well.*

(*iii*) *Term sign: Always positive for an admissible tree. For an admissible tree-pair, the sign is determined by the* Sign Determination Algorithm.

Remark 1 *It should be noted that CC and VS edges (those edges that* must appear*) would never form a loop; otherwise, the network would be pathological, hence of no practical importance.*

Theorem 1 (Main Theorem) *Under Assumption 1, all admissible terms defined by Definition 2 sum up to zero.*

B. Proof of the Main Theorem

The proof is quite delicate. Only the key steps are sketched for the limited space.

This proof follows an algebraic approach based on tableau formulation and an application of the *Binet-Cauchy Theorem* [24]. Several new techniques are introduced in the proof.

First an ε symbol is introduced in place of some zero elements in the tableau matrix. The tableau matrix is formed by stamping all circuit elements together. The ε symbol help use trace the symbols associated with the dependent sources.

The second technique is element stamping. The branch equations of a circuit (converted to a graph) can be written in matrix form

$$\mathbb{Z}I + \mathbb{Y}U = 0, \tag{3}$$

where I and U are the branch current and voltage vectors, respectively. Each branch equation takes one of the following forms

$$\begin{array}{lll} U_i = Z_i I_i & & \text{(Z edge)} \\ I_i = Y_i U_i & & \text{(Y edge)} \\ U_i = E_{i,j} U_j, & I_j = 0 & \text{(VCVS)} \\ I_i = F_{i,j} I_j, & U_j = 0 & \text{(CCCS)} \\ I_i = G_{i,j} U_j, & I_j = 0 & \text{(VCCS)} \\ U_i = H_{i,j} I_j, & U_j = 0 & \text{(CCVS)} \\ U_j = 0, & I_j = 0 & \text{(NU edge).} \end{array} \tag{4}$$

Treating all Z (impedance) edges as Y (admittance) edges simplifies the algebraic manipulation. Thus the branch equation for a Z edge in (4) will be rewritten as $I_i = Z_i^{-1} U_i$, i.e. all *impedances* are treated as *admittance* in the graph.

The appearance of the preceding seven types of elements in the matrices $\mathbb{Z}$ and $\mathbb{Y}$ can be described by the stamps listed below (only a few are listed to save space):

$$\begin{array}{ccc} \text{Stamp in } \mathbb{Z} & \text{Stamp in } \mathbb{Y} & \\ \begin{bmatrix} \ddots & & \\ & 1 & \\ & & \ddots \end{bmatrix} & \begin{bmatrix} \ddots & & \\ & -Y_i & \\ & & \ddots \end{bmatrix} & (Y) \\ \begin{bmatrix} \varepsilon & \cdots & H_{i,j} \\ & \ddots & \vdots \\ & & \varepsilon \end{bmatrix} & \begin{bmatrix} -1 & & \\ & \ddots & \\ & & -1 \end{bmatrix} & (CCVS) \\ \begin{bmatrix} \varepsilon & & \\ & \ddots & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 0 & \cdots & 1 \\ & \ddots & \vdots \\ & & 0 \end{bmatrix} & (Nullor) \end{array} \tag{5}$$

Note that we wrote the stamps for the dependent sources in upper triangular form just for convenience. Also note that we have intentionally placed all the minus signs to the diagonal elements of matrix $\mathbb{Y}$ and replaced all the *zeros* on the diagonal of matrix $\mathbb{Z}$ by ε, making the matrix $\mathbb{Z}$ is nonsingular for $\varepsilon > 0$. Eventually the original circuit equations will be recovered by taking the limit $\varepsilon \to 0$.

Matrix $\mathbb{Z}$ is diagonalized for easy manipulation later on. First, the nonzero off-diagonals in matrix $\mathbb{Z}$ coming with the stamps for CCCS and CCVS can be eliminated by row transformations. Note that the presence of ε symbols makes this operation possible.

The key part of the proof is algebraic manipulation. Given a connected network, after removing a spanning tree from the network, the remaining branches form a subgraph called *cotree*. Let I_t (resp. U_t) and I_c (resp. U_c)

be the vectors containing branch currents (resp. voltages) corresponding to the edges on the tree and cotree, respectively. By tableau formulation, the network equation can be written as

$$\begin{bmatrix} A_t & & & A_c \\ & B_c & B_t & \\ Z_{(1)} & Y_{(2)} & Y_{(1)} & Z_{(2)} \\ Z_{(3)} & Y_{(4)} & Y_{(3)} & Z_{(4)} \end{bmatrix} \begin{bmatrix} I_t \\ U_c \\ U_t \\ I_c \end{bmatrix} = 0 \tag{6}$$

where $A := \begin{bmatrix} A_t & A_c \end{bmatrix}$ is the *reduced incidence matrix* with A_t corresponding to a preselected tree, and $B := \begin{bmatrix} B_c & B_t \end{bmatrix}$ is the *fundamental loop matrix* with B_c corresponding to the cotree (see the textbook [12]). An empty block in (6) indicates all-zero entries. Note that $\det|A_t| = \pm 1$ and $\det|B_c| = \pm 1$.

After some algebraic manipulation, the determinant of the coefficient matrix of (6) becomes

$$\det|\mathbb{M}| = \det|A_t| \ \det|B_c| \ \det|\hat{Z}_{(1)}| \ \det|\hat{Z}_{(4)}| \times \\ \times \begin{vmatrix} I & & & Q \\ & I & -Q^{\mathrm{T}} & \\ I & \hat{Z}_{(1)}^{-1}\widetilde{Y}_{(2)} & \hat{Z}_{(1)}^{-1}\widetilde{Y}_{(1)} & \\ & \hat{Z}_{(4)}^{-1}\widetilde{Y}_{(4)} & \hat{Z}_{(4)}^{-1}\widetilde{Y}_{(3)} & I \end{vmatrix}, \tag{7}$$

where $Q = A_t^{-1}A_c$ and $B_c^{-1}B_t = -Q^{\mathrm{T}}$ (see Corollary 2.8 in [12]). Let $\mathbb{M}_1$ be the matrix in the last determinant of (7). Applying some algebraic identities, we obtain that

$$\det|\mathbb{M}_1| = \left| \begin{bmatrix} A_t & A_c \end{bmatrix} \begin{bmatrix} \hat{Z}_{(1)}^{-1} & \\ & \hat{Z}_{(4)}^{-1} \end{bmatrix} \begin{bmatrix} \widetilde{Y}_{(1)} & \widetilde{Y}_{(2)} \\ \widetilde{Y}_{(3)} & \widetilde{Y}_{(4)} \end{bmatrix} \begin{bmatrix} A_t^{\mathrm{T}} \\ A_c^{\mathrm{T}} \end{bmatrix} \right| \tag{8}$$

Since $|A_t||B_c| = \pm 1$, the identity of $\det|\mathbb{M}| = 0$ is equivalent to (by (7))

$$\det|\hat{Z}_{(1)}| \ \det|\hat{Z}_{(4)}| \det|\mathbb{M}_1| = 0.$$

Let

$$\hat{\mathbb{Z}} = \hat{\mathbb{Z}}(\varepsilon) := \begin{bmatrix} \hat{Z}_{(1)}(\varepsilon) & \\ & \hat{Z}_{(4)}(\varepsilon) \end{bmatrix}$$

and

$$\mathcal{S} = \left|\hat{\mathbb{Z}}(\varepsilon)\right| \left| A\hat{\mathbb{Z}}^{-1}(\varepsilon)\widetilde{\mathbb{Y}}(\varepsilon)A^{\mathrm{T}} \right|. \tag{9}$$

It is clear that $\det|\mathbb{M}| = 0$ is equivalent to $\mathcal{S} = 0$.

By Binet-Cauchy Theorem [24] and noticing that $\hat{Z}$ is diagonal, we have

$$\begin{aligned} \mathcal{S} &= \left|\hat{\mathbb{Z}}\right| \sum_{j_1,\cdots,j_n} \left[A\hat{\mathbb{Z}}^{-1}\right] \begin{pmatrix} 1 & \cdots & n \\ j_1 & \cdots & j_n \end{pmatrix} \left[\widetilde{\mathbb{Y}}A^{\mathrm{T}}\right] \begin{pmatrix} j_1 & \cdots & j_n \\ 1 & \cdots & n \end{pmatrix} \\ &= \left|\hat{\mathbb{Z}}\right| \sum_{\substack{j_1,\cdots,j_n \\ k_1,\cdots,k_n}} \left(\prod_{i=1}^{n} \hat{Z}_{j_i}^{-1} \right) A \begin{pmatrix} 1 \cdots n \\ j_1 \cdots j_n \end{pmatrix} \widetilde{\mathbb{Y}} \begin{pmatrix} j_1 \cdots j_n \\ k_1 \cdots k_n \end{pmatrix} A^{\mathrm{T}} \begin{pmatrix} k_1 \cdots k_n \\ 1 \cdots n \end{pmatrix} \end{aligned} \tag{10}$$

where n is the number of nodes in the network (excluding the ground node) and the notation $A\begin{pmatrix} 1 & \cdots & n \\ j_1 & \cdots & j_n \end{pmatrix}$ denotes the major of matrix A formed by the rows $\{1, \cdots, n\}$ and the columns $\{j_1, \cdots, j_n\}$. By default the index set $\{j_1, \cdots, j_n\}$ for summation means the summation over all distinct indices satisfying $j_1 < j_2 < \cdots < j_n$.

We observe from (10) that a nonzero term in the summand results from the condition that all the three factors

$$A\begin{pmatrix} 1 & \cdots & n \\ j_1 & \cdots & j_n \end{pmatrix}, \quad \widetilde{\mathbb{Y}}\begin{pmatrix} j_1 & \cdots & j_n \\ k_1 & \cdots & k_n \end{pmatrix}, \quad A^{\mathrm{T}}\begin{pmatrix} k_1 & \cdots & k_n \\ 1 & \cdots & n \end{pmatrix}$$

are nonzero. It is a well-known fact that a nonzero major of the reduced incidence matrix corresponds to a tree. Since A is a reduced incidence matrix, if the index sets $\{j_1, \cdots, j_n\}$ and $\{k_1, \cdots, k_n\}$ coincide, then the corresponding term is determined by a *tree* with its edges identified by the index set, while if the index sets are nonidentical, then the corresponding term is determined by a *tree-pair*. Note that in the case of tree-pair, the row indices of matrix $\widetilde{\mathbb{Y}}$ associated with the term are the edge numbers of the left-tree (*L-tree*), while the column indices of matrix $\widetilde{\mathbb{Y}}$ associated with the term are the edge numbers of the right-tree (*R-tree*).

However, given any spanning tree or tree-pair from the circuit graph, the resulting term could be zero if the determinant represented by $\widetilde{\mathbb{Y}}\begin{pmatrix} j_1 & \cdots & j_n \\ k_1 & \cdots & k_n \end{pmatrix}$ vanishes. To prevent from enumerating trees or tree-pairs that result in vanishing terms, one needs to identify constraints on the tree edges that lead to admissible tree-pairs (or trees). The constraints are actually defined by the structure of matrices $\widetilde{\mathbb{Y}}$ and $\hat{\mathbb{Z}}$ that determines whether the resulting term vanishes.

Conditions stated in Definition 1 can be derived from analyzing the special structure of matrices $\widetilde{\mathbb{Y}}$ and $\hat{\mathbb{Z}}$, taking into account of the role of ε (which is vanishing). The details of the rest of derivation is omitted.

References

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

[2] ——, "Topological analysis of linear nonreciprocal networks," *Proc. IRE*, vol. 45, pp. 829–838, June 1957.

[3] S. Mason and H. Zimmermann, *Electronic Circuits, Signals, and Systems*. New York: Wiley, 1960.

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

[5] W. Mayeda, *Graph Theory*. New York: Wiley-Interscience, 1972.

[6] W. Chen, "Topological analysis for active networks," *IEEE Trans. on Circuit Theory*, vol. CT-12, pp. 85–91, 1965.

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

[8] S.-D. Shieu and S.-P. Chan, "Topological formulation of symbolic network functions and sensitivity analysis of active networks," *IEEE Trans. on Circuits and Systems*, vol. CAS-21, no. 1, pp. 39–45, 1974.

[9] P. Lin and G. Alderson, "Computer generation of symbolic network functions – a new theory and implementation," *IEEE Trans. on Circuit Theory*, vol. CT-20, pp. 48–56, 1973.

[10] P. Sannuti and N. Puri, "Symbolic network analysis – An algebraic formulation," *IEEE Trans. on Circuits and Systems*, vol. CAS-27, no. 8, pp. 679–687, 1980.

[11] P. Lin, "A survey of applications of symbolic network functions," *IEEE Trans. on Circuit Theory*, vol. CT-20, no. 11, pp. 732–737, 1973.

[12] ——, *Symbolic Network Analysis*. New York: Elsevier, 1991.

[13] G. Gielen and W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. Norwell, MA: Kluwer, 1991.

[14] F. Fernández, A. Rodríguez-Vázquez, J. Huertas, and G. Gielen, *Symbolic Analysis Techniques – Applications to Analog Design Automation*. New York: IEEE Press, 1998.

[15] 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.

[16] M. Hassoun and P. Lin, "A hierarchical network approach to symbolic analysis of large-scale networks," *IEEE Trans. on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 42, no. 2, pp. 201–211, 1995.

[17] 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.

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

[19] 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.

[20] 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.

[21] X. Li and Z. Yin, "The algorithm and program scheme to find out all valid trees and valid tree-pairs," in *Proc. 5th Int'l Conference on ASIC*, Beijing, China, 2003, pp. 298–301.

[22] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a BDD package," in *Proc. 27th ACM/IEEE Design Automation Conference*, Orlando, FL, 1990, pp. 40–45.

[23] W. Chen and G. Shi, "Implementation of a symbolic circuit simulator for topological network analysis," in *Proc. Asia Pacific Conference on Circuits and Systems (APCCAS)*, Singapore, Dec. 2006.

[24] W. Chen, *Applied Graph Theory – Graphs and Electrical Networks*. Amsterdam: North-Holland, 1976.