Approximate Symbolic Analysis of Large Analog Integrated Circuits

Qicheng Yu and Carl Sechen

Department of Electrical Engineering, FT-10
University of Washington
Seattle, WA 98195

Abstract

This paper describes a unified approach to the approximate symbolic analysis of large linearized analog circuits. It combines two new approximation-during-computation strategies with a variation of the classical two-graph tree enumeration method. The first strategy is to generate common trees of the two-graphs, and therefore the product terms in the symbolic network function, in the decreasing order of magnitude. The second approximation strategy is the sensitivity-based simplification of two-graphs, which excludes from the two-graphs many circuit elements that have little effect on the network function being derived. Our approach is therefore able to symbolically analyze much larger analog integrated circuits than previous reported, using complete small signal models for the semiconductor devices. We show accurate yet reasonably sized symbolic network functions for integrated circuits with up to 39 transistors whereas previous approaches were limited to less than 15.

1. Introduction

Symbolic analysis provides insight into circuit behavior that numerical simulation does not. Symbolic analysis as an important tool for the design of analog integrated circuits is gaining wider recognition, as evidenced by the success of several symbolic simulators [9][20][16][6][12]. Such success is driven by the strong need to reduce the long design time and high design cost of analog IC's as compared to their digital counterparts of the same area and is based on the increasing power of computers and symbolic analysis algorithms [10]. However, the size of circuits that can be symbolically analyzed is still small by industrial standards. The difficulty, which is imposed by the exponential growth of product terms in an expanded symbolic network function with respect to the circuit size [17], is only partially overcome by various approximation strategies [9][7][2][20][1] and decomposition strategies [21][12][13] employed up to now.

Our new approach presented in this paper efficiently analyzes larger analog integrated circuits than those previously handled. Our approach combines two new approximation-during-computation strategies with a variation of the classical two-graph tree enumeration method. The first strategy generates common trees (tree admittance products) of the two-graphs in decreasing order of magnitude. This approach avoids the burden of computing all the product terms only to find most of them numerically negligible, and is accomplished by our algorithm for generating color-constrained spanning trees in the order of weight. The second approximation strategy is the sensitivity-based simplification of two-graphs which excludes many circuit elements that have little effect on the network function and significantly reduces the complexity of the two-graphs before tree enumeration. Our approach has been implemented and it efficiently generates symbolic network functions of reasonable size and accuracy in expanded format for integrated circuits with up to 39 transistors, using complete small signal models for the devices. Such circuit examples are much larger than previously reported in the literature. For even larger circuits, the limit is imposed mainly by the interpretability of the generated symbolic expression.

The rest of the paper is organized as follows: Section 2 describes symbolic analysis background. Section 3 is dedicated to the two-graph tree-enumeration method. Section 4 shows how to generate only the important product terms in the expanded format and Section 5 introduces the second approximation strategy of sensitivity-based two-graph simplification. Section 6 presents the error control mechanism for tree generation and Section 7 gives experimental results obtained from our symbolic analysis program. Section 8 is the conclusion.

2. Symbolic Circuit Analysis Background

Symbolic analysis produces various network functions in symbolic form such that circuit element parameters are kept as symbols. It is usually done in the complex frequency domain for linearized analog circuits. For continuous-time circuits, an arbitrary network function can be expressed in the *expanded format* as the ratio of two polynomials in the Laplace variable s:

$$T(s) = \frac{N(s)}{D(s)} = \frac{\sum_j b_j s^j}{\sum_i a_i s^i} = \frac{b_0 + b_1 s^1 + \ldots + b_n s^n}{a_0 + a_1 s^1 + \ldots + a_d s^d}, \qquad (1)$$

where a_i and b_j are sum-of-product expressions in terms of circuit element parameters.

The main difficulty of symbolic analysis is the exponential growth of product terms with the number of nodes and elements in a circuit. As a result, exact symbolic expressions can only be obtained for small circuits. Two strategies, approximation and decomposition, have been adopted to tackle this problem.

We can approximate an exact symbolic expression by removing unimportant symbols or terms. For an expanded format, product terms with small magnitudes are deleted or not generated in both the denominator and the numerator. Nominal values of symbols must be known for this purpose. Decomposition carries out symbolic expression generation in a divide-and-conquer fashion. Either the circuit or the associated graphs can be decomposed into smaller blocks, and the analysis results of these blocks

pieced together. In this manner, nested expressions or sequences of expressions can be produced for large circuits. Approximation is a must for symbolic analysis of even moderate-sized circuits, and for still larger circuits, both approximation and decomposition are necessary.

Even with approximation and decomposition, the limit to symbolic analysis is twofold. They are *generatability* and *interpretability* of symbolic network functions. When a circuit is too large, generation of any reasonably accurate symbolic network function is simply beyond the ability of currently available algorithms. Furthermore, not all generated symbolic network functions are directly interpretable by a user because of their large size or complicated format. With the expansion in the power of algorithms, this limit of interpretability will outweigh generatability and become the new bottleneck for symbolic analysis.

Approximation strategies fall into two classes: *approximation-during-computation* (ADC) and *approximation-after-computation* (AAC) [2]. AAC generates the exact symbolic expression and then operates on it to arrive at a compact approximate expression, while ADC arrives at the approximate expression without explicit knowledge of the exact expression. As a result, ADC expands generatability as well as interpretability while AAC only expands interpretability. Although ADC is superior, it is also more difficult. One has to eliminate symbols or terms before they explicitly enter the symbolic expression. With such techniques, published circuit examples of generatability for integrated circuits are less than 15 transistors, aside from [14] and [15]. The approach in [14] and [15], however, has the following limitations: 1) The symbolic expressions produced are in a nested format with embedded term cancellations; 2) If the nested expression is translated into an irreducible expanded format, it may include alien product terms created in the device elimination process that have no origin in the exact unsimplified symbolic expression.

Decomposition may be the only hope for the symbolic analysis of even larger circuits. The core of a decomposition strategy is how the partitioning is done. Typical transistor circuits unfortunately do not possess the structural regularity or loose coupling characteristics that existing decomposition schemes require. Again, no circuit example of applying decomposition to a large integrated circuit has been reported.

3. The Two-Graph Tree Enumeration Method

Our approach starts with the two-graph tree enumeration method of symbolic analysis, which is mainly due to Mayeda and Seshu [18] and is so called because a voltage graph G_V and a current graph G_I are always used. The original results could only handle one type of active element, the voltage controlled current source (VCCS), but the method was further developed by many researchers for general linear networks to include virtually all active elements. However, these results are generally too complicated to implement [17]. Based on the same underlying concepts, we have generated a set of specific rules for constructing and using the two graphs from the tableau approach of circuit equation formulation [4], which is applicable to a linear circuit encompassing the following types of elements:

- Resistors and capacitors;
- Voltage controlled current sources and current controlled voltage sources (CCVSs);
- Nullors (for ideal opamps operating in the linear mode);
- Opens and shorts;
- Independent current and voltage sources.

These elements share the property that the non-trivial part of their branch voltage-current relations (BVCRs) can all be represented by admittances. For example, the BCVR for a CCVS can be written as

$$v_1 = 0 \text{ and } i_1 = g_h v_2 \tag{2}$$

instead of

$$v_1 = 0 \text{ and } v_2 = r_h i_1. \tag{3}$$

This property allows them to be treated uniformly and simplifies the topological rules for tree enumeration. Voltage-controlled voltage sources (VCVSs) and current-controlled current sources (CCCSs) can be modeled by VCCSs and CCVSs if desired.

We represent all resistors, capacitors, VCCSs and CCVSs as admittances, which will be referred to as *elements*, or *edges* when they appear in associated graphs. For resistors, VCCSs and CCVSs $y = g$, and for capacitors $y = sC$. Then the following lemma is in order [4][17]:

Lemma 1. *If every admittance type element of a circuit consisting of the above mentioned types of elements is given a distinct symbol for its admittance (i.e., no matching conditions are considered) and if the unsimplified symbolic expressions of the numerator or denominator of an arbitrary network function are expressed in the irreducible expanded format of sum of element admittance products, then both expressions are polynomials of degree one in each element admittance and every product term (with its distinct element admittance combination) has a numeric coefficient of either* 1 *or* -1. *Here irreducible means no two product terms in the expression share the same element admittance combination.*

Such a network function can therefore also be written as:

$$T = \frac{N}{D} = \frac{\sum_j \left(\pm \prod_{l=1}^{L} y_{jl} \right)}{\sum_i \left(\pm \prod_{k=1}^{K} y_{ik} \right)}, \tag{4}$$

where K and L are the numbers of admittances in the product terms of the denominator and the numerator, respectively.

The two-graph tree enumeration method for the above mentioned class of circuits can be described as follows: For either the numerator $N(s)$ or the denominator $D(s)$ of a network function, which is generically denoted as

$$P(s) = \sum_{i=0}^{Np} p_i s^i, \tag{5}$$

where N_p is the highest power of s and $p_i s^i$ is the sum of product terms containing i capacitors, we can construct a voltage graph G_V (v-graph) and a current graph G_I (i-graph), which are connected undirected graphs. They have the same number of vertices and the same edge set, but the vertices each edge connects to may be different in one versus the other. Each edge corresponds to an admittance type element in the circuit and is labeled with that admittance.

The structures of G_V and G_I are such that the product terms in $P(s)$ correspond one-to-one to the common spanning trees of G_V and G_I. Each product term equals the product of edge admit-

tances in the common spanning tree and is termed a *tree admittance product*. A common spanning tree, or *common tree* for short, is a set of edges that constitute a spanning tree in both G_V and G_I. The structures of G_V and G_I are based on the structure of the circuit and on the specific circuit variable which $P(s)$ represents (*e.g.*, node voltage of node 1 or branch current of resistor r_1). The combination of G_V and G_I is called a *two-graph*. The problem of finding all the product terms in the irreducible expression is therefore converted to the problem of finding all the common spanning trees of the two-graph. Two-graph tree enumeration has the *cancellation free* property, as long as no matching conditions are considered.

Rules to construct G_V and G_I are specified with respect to every type of circuit element and to the circuit variable associated with $P(s)$. (The rules and their derivation are not included for brevity. A similar treatment of the class of circuits containing VCCSs as the only type of active elements can be found in [18].) Each product term has an associated sign. The rules given by Mayeda and Seshu [18] to determine the sign were extended for our application. Such rules are efficient: Given a common tree, the sign of the tree admittance product can be determined by performing a depth-first search on both the *v*-graph tree and the *i*-graph tree.

All previous implementations of two-graph tree enumeration attempt to generate all the common trees. This severely limits the size of circuit that can be handled because the number of trees grows exponentially with the circuit size.

4. Generation of Significant Common Trees [1]

In typical integrated circuits, small signal circuit element values often vary by several orders of magnitude. Therefore it is very likely that the numerator or denominator is dominated by a small portion of all the product terms in the irreducible expanded expression [11]. In order for the tree enumeration technique to be applicable to large circuits, we seek to generate only those common trees whose tree admittance product magnitude is not negligible compared to the magnitude of $P(s)$, called *significant common trees*. One method is to generate the spanning trees of G_V in decreasing order of the magnitude of their tree admittance product, and then check whether they are also spanning trees of G_I.

In combinatorial optimization there are efficient algorithms for generating the *K* lowest weight spanning trees of a weighted undirected graph in increasing order of weight, in which the weight of a tree is defined as the sum of the edge weights [8][25]. We therefore define the weights for the edges in G_V and G_I so that a spanning tree with the largest magnitude of tree admittance product becomes a spanning tree with lowest weight. The translation from edge admittance to weight is given by:

$$\begin{array}{cc} resistors, VCCSs, CCVSs & g \Rightarrow -\log|g| \\ capacitors & sC \Rightarrow -\log|sC| \end{array} \tag{6}$$

Here $|x|$ denotes the magnitude of x. A spanning tree whose tree admittance product is

$$g_{l_1} g_{l_2} \ldots g_{l_{d-i}} sC_{l_{d-i+1}} sC_{l_{d-i+2}} \ldots sC_{l_d} \tag{7}$$

has a weight of

$$\begin{aligned} &-\log|g_{l_1}| - \log|g_{l_2}| - \ldots - \log|g_{l_{d-i}}| \\ &-\log|sC_{l_{d-i+1}}| - \log|sC_{l_{d-i+2}}| - \ldots - \log|sC_{l_d}| \end{aligned} \tag{8}$$

after translation.

Using a *K* lowest weight spanning tree algorithm on G_V, we can generate its spanning trees in decreasing order of the magnitude of their tree admittance product. One such algorithm by Gabow [8] has a time complexity of $O(E \log E + KE\alpha(E, V))$, where V is the number of vertices, E is the number of edges in the graph, K is the number of spanning trees generated and $\alpha(E, V)$ is the very slow growing inverse of Ackermann's function [5].

Gabow's algorithm achieves efficiency by making use of a fundamental property of the spanning trees of an undirected graph. The property is that if the k lowest weight spanning trees have been found, then the $(k+1)$-st lowest weight spanning tree can be derived from at least one of the k lowest weight spanning trees by exchanging an edge in the tree with an edge not in the tree. This means the search for the next tree is limited to the "neighborhood" of those trees already generated.

For a given frequency f ($s = j2\pi f$), we can apply Gabow's algorithm to G_V, and every spanning tree of G_V so generated is checked against G_I. If it is a spanning tree of G_I also, then it is a valid product term in $P(s)$; otherwise, it is ignored. Each spanning tree of G_V takes $O(E\alpha(E, V))$ time to generate and $O(V)$ time to be checked against G_I. If it is a common tree, $O(V)$ additional time is needed to determine the sign of the tree admittance product. In this manner, we can efficiently generate the product terms in $P(s)$, in decreasing order of magnitude until the generated set of product terms well approximates $P(s)$. This procedure is carried out for both the numerator and denominator so an approximate symbolic expression for the desired network function $T(s)$ results.

Symbolic expressions so generated are valid only in the vicinity of a given frequency, however. For product terms in $p_i s^i$ of the following form:

$$\pm g_{l_1} g_{l_2} \ldots g_{l_{d-i}} C_{l_{d-i+1}} C_{l_{d-i+2}} \ldots C_{l_d} s^i, \tag{9}$$

when the frequency changes, the relative magnitudes of product terms with different powers of *s* change. Hence a product term discarded at one frequency may become significant at another frequency.

This problem can be avoided by generating product terms with different powers of *s* separately. That is, we seek to generate spanning trees of G_V with exactly *i* capacitor edges in decreasing order of the magnitude of tree admittance product, and iterate over all *i*. The combinatorial optimization problem so formed is: Given a weighted undirected graph where the edges are colored either red or green, efficiently generate K_i lowest weight spanning trees with exactly *i* red edges in increasing order of weight for all feasible *i*.

The problem is different from the uncolored version in one aspect. When the k lowest weight spanning trees with exactly *i* red edges have been found, the $(k+1)$-st such spanning tree may not be derivable from any one of them by exchanging one edge in the tree with one edge not in the tree. So for the color-constrained problem, the search space for the next spanning tree in the sequence must expanded. By expanding this search space, we have obtained an efficient algorithm for the problem [25]. The

1. The approximation strategy in this section was first proposed by the authors in [25]. Subsequently and independently, a similar strategy was reported by Wambacq et al. in [23].

K_i lowest weight spanning trees with exactly i red edges can be generated in order in $O(K_i E\alpha(E,V))$ time from a lowest weight spanning tree with exactly i red edges. Thus each spanning tree with color constraints takes $O(E\alpha(E,V))$ time to generate. The modified algorithm is applied to G_V and the generated spanning trees are checked against G_I. If we adopt the error control scheme used in [22] and [20], this procedure continues until the sum of the tree admittance products well approximates $p_i s^i$. When the same has been done for every $p_i s^i$, $i = 1, 2, \ldots, N_p$, an approximate symbolic expression for $P(s)$ is obtained. The combination of expressions for $N(s)$ and $D(s)$ gives an approximate symbolic expression for $T(s)$ which is valid for the entire frequency range of interest.

For our algorithm, if K is the total number of spanning trees generated for G_V of $P(s)$,

$$K = \sum_i K_i, \tag{10}$$

then the total time to compute the approximate symbolic expression for $P(s)$ is $O(E\log E + K E\alpha(E,V))$.

The ratio of the number of *v*-graph spanning trees generated over the number of common spanning trees among them largely determines the overall efficiency of the tree enumeration process. This ratio varies with the size and structure of the two-graph. Nevertheless, the significant savings from not having to generate every valid product term makes the symbolic analysis of large circuits viable.

5. Sensitivity-Based Two-Graph Simplification

Our experience with generating significant common trees shows that the number of such common trees and the number of *v*-graph trees needed to obtain good approximation accuracy grow fast with the size of the two-graph. For improved interpretability of the approximate expression and for tree generation efficiency, we incorporate a *sensitivity-based simplification* scheme for the two-graphs. With this scheme, we reduce the number of edges, each of which corresponds to a unique circuit element, and the number of vertices in the two-graph.

If we define

$$N(y_1, y_2, \ldots, y_p; y_{p+1}, y_{p+2}, \ldots, y_{p+q}) \tag{11}$$

to be the sum of product terms in N that contain $y_1, y_2, \ldots, y_p$ and do not contain $y_{p+1}, y_{p+2}, \ldots, y_{p+q}$ divided by $y_1 y_2 \ldots y_p$ and define

$$D(y_1, y_2, \ldots, y_p; y_{p+1}, y_{p+2}, \ldots, y_{p+q}) \tag{12}$$

similarly, then for a specific element y, the network function T can be written as

$$T = \frac{N}{D} = \frac{N(\ ;y) + yN(y;\ )}{D(\ ;y) + yD(y;\ )}. \tag{13}$$

The simplification scheme consists of two stages. We first treat the two-graphs for N and D individually, then treat both two-graphs together. When certain conditions have to be met over all frequencies, we test them over a set of *sample frequencies* chosen from the frequency range of interest.

5.1 Simplifying Individual Two-Graphs

In the first stage, we define

$$c(y,N) = \frac{yN(y;\ )}{N} \text{ and } \bar{c}(y,N) = \frac{N(\ ;y)}{N}, \tag{14}$$

which are usually complex numbers for a given frequency f ($s = j2\pi f$), to be the *contribution* and *contribution complement* of y to N. (A technique of efficiently computing them can be found in [3].) If the magnitude of $c(y,N)$ or $\bar{c}(y,N)$ is small over all frequencies, we may remove $yN(y;\ )$ or $N(\ ;y)$, *i.e.*, the group of product terms containing or not containing y, from N without introducing much numerical error. We call such y whose $c(y,N)$ or $\bar{c}(y,N)$ is small over all frequencies *weakly contributing* or *strongly contributing* elements for N, respectively. The removal of $yN(y;\ )$ or $N(\ ;y)$ is equivalent to deleting or contracting, respectively, the edge y in both *i*-graph and *v*-graph before tree enumeration.

However, because we are removing a group of product terms containing or not containing y, it is possible that two different product terms with similarly large magnitudes (as compared to N) but opposite signs, called *numerically cancelling large product terms*, are removed together. In such cases, the information content of N is distorted.

The most important information provided by a symbolic network function is how the elements contribute, directly or through interaction with each other, to the network function. Consequently a good way to preserve the information is to limit the error in the *contribution* of an element to both N and D when the symbolic network function T is simplified. We attempt to remove (delete or contract in the two-graph) each weakly contributing or strongly contributing element y from N in turn, and after each attempt we check the contribution of every remaining element to N. If a remaining element y' has a new contribution to N that is significantly different from the original value of $c(y',N)$ prior to all such element removals at some frequency, then this attempt to remove y is rejected. Although such a protection mechanism does not guarantee against the removal of numerically cancelling large product terms, it greatly reduces its probability. The element selected for the next removal attempt is one that will cause the least cumulative error in N. Meanwhile, we limit the cumulative error in N incurred during all removals. D is processed similarly. After the first stage we call the network function represented by the simplified two-graphs T_1:

$$T_1 = \frac{N_1}{D_1}. \tag{15}$$

Example 1. Consider $N = g_2 + g_m$ where $g_2 = 1$ and $g_m = 100$. Since $c(g_2,N) \approx 0.01$ and $\bar{c}(g_m,N) \approx 0.01$, g_2 is a weakly contributing element and g_m is a strongly contributing element. Thus we may try deleting the edge for g_2 or contracting the edge for g_m. If we choose to contract the edge for g_m, g_m becomes an independent factor.

Consider $D = g_1g_2 + g_1g_3 + g_2g_3 - g_mg_2$ where $g_1 = g_2 = g_3 = 1$, $g_m = 2$. Since $c(g_2,D) = 0$ we may try deleting the edge for g_2. If we do, the contribution of g_1 changes from $c(g_1,D) = 2$ to $c(g_1, D(\ ;g_2)\ ) = 1$. This is a significant change, so the deletion of the edge for g_2 is rejected. The two-graph for D can not be simplified in this stage.

5.2 Simplifying Both Two-Graphs Simultaneously

In the second stage we further simplify the numerator and denominator two-graphs by deleting or contracting certain edges that are common to both and whose *contribution* to both are almost equal for all frequencies. The effect of deleting or contracting a common edge y on the network function is to remove

the group of product terms not containing or containing y simultaneously from N_1 and D_1, respectively. The rationale behind such removal is that the large-change sensitivity of the network function with respect to some of the circuit elements is small over all frequencies in the range of interest.

We note that

$$T_1(\ ;y) = \frac{N_1(\ ;y)}{D_1(\ ;y)} \tag{16}$$

and

$$T_1(y;\ ) = \frac{yN_1(y;\ )}{yD_1(y;\ )} = \frac{N_1(y;\ )}{D_1(y;\ )} \tag{17}$$

are the network functions represented by the two-graphs derived from those for N_1 and D_1 by deleting and contracting y in both two-graphs, respectively. Define

$$e_c(y;T_1) = \left(\frac{N_1(y;\ )}{D_1(y;\ )} - \frac{N_1}{D_1}\right) \Big/ \left(\frac{N_1}{D_1}\right) \tag{18}$$

and

$$e_d(y;T_1) = \left(\frac{N_1(\ ;y)}{D_1(\ ;y)} - \frac{N_1}{D_1}\right) \Big/ \left(\frac{N_1}{D_1}\right) \tag{19}$$

as the *dual contraction error* and *dual deletion error* of T_1 with respect to y, which are generally complex numbers. If the magnitude of $e_c(y;T_1)$ or $e_d(y;T_1)$ is small over all frequencies, we can contract or delete y from both the numerator and the denominator two-graphs without causing much numerical error in T_1. Such an operation is called a *dual contraction* or a *dual deletion* and such an edge is a *contraction candidate* or a *deletion candidate*, respectively. A common edge can be a contraction candidate and a deletion candidate simultaneously.

Again, consider the impact of a dual contraction or deletion on the information content of T_1. Here we wish to limit the error in the *contribution* of every remaining element to both N_1 and D_1. For each contraction candidate y, we attempt a dual contraction and then check the contribution of all remaining elements to N_1 and D_1. For the dual contraction to be acceptable to a remaining element y', either the contribution of y' to N_1 and D_1 does not change significantly compared to before all dual contractions and deletions, or, when y' is common to the numerator and the denominator and is a deletion candidate, the contribution of y' to both N_1 and D_1 becomes very small. The latter case usually occurs when a deletion candidate edge becomes a self-loop in both two-graphs because of previous contractions. In the former case the dual contraction is simply accepted by y'. In the latter case the dual contraction is accepted by y' and y' is immediately dual-deleted. Such dual deletions are termed *derivative dual deletions*. If not all of the y''s accept the dual contraction, the dual contraction is rejected; otherwise we proceed to check the acceptability of all associated derivative dual deletions.

Likewise for the derivative dual deletions to be acceptable to a remaining element y'', the contribution of y'' to N_1 and D_1 should not change significantly compared to before all dual contractions and deletions. If every y'' accepts the dual deletions, then the derivative dual deletions together with the prior dual contraction are all accepted; otherwise, they are all rejected. In either case we advance to the next dual contraction attempt.

Finally for each remaining deletion candidate we attempt a dual deletion with similar acceptance criteria.

At every step the next candidate selected for a dual contraction or deletion attempt is one that will cause the least cumulative error in T_1. In the whole process, we limit the cumulative error in T_1 incurred during all the dual contractions and deletions.

After the second stage of sensitivity-based simplification, we call the network function represented by the simplified two-graphs T_2:

$$T_2 = \frac{N_2}{D_2}. \tag{20}$$

Example 2. Consider

$$T_1 = \frac{g_3 + g_4}{(g_1 + g_2)\ (g_3 + g_4)} \tag{21}$$

where $g_1 = g_2 = g_3 = g_4 = 1$. We have $e_c(g_3, T_1) = 0$, $e_c(g_4, T_1) = 0$, $e_d(g_3, T_1) = 0$ and $e_d(g_4, T_1) = 0$. So g_3 and g_4 are both contraction candidates and both deletion candidates. We attempt to dual-contract the edge for g_3 so that

$$T_1(g_3;\ ) = \frac{1}{g_1 + g_2}. \tag{22}$$

Since g_4 is a deletion candidate and the new contribution $c(g_4, N_1(g_3;\ )) = 0$ and $c(g_4, D_1(g_3;\ )) = 0$ are "very small", we derivatively dual-delete the edge for g_4. (In fact, it has already been deleted because it has become a self-loop.) We then check the contribution of the remaining elements in $N_1(g_3;g_4)$ and $D_1(g_3;g_4)$. Since

$$c(g_1, D_1(g_3;g_4)\ ) = c(g_1, D_1) = 0.5 \tag{23}$$

and

$$c(g_2, D_1(g_3;g_4)\ ) = (c(g_2, D_1)) = 0.5 \tag{24}$$

does not change after the dual removals, the dual-removals are accepted.

6. Error Control Mechanism

The approximation error in our approach originates from sensitivity-based two-graph simplification and significant common tree generation. The approximation error associated with sensitivity-based two-graph simplification is controlled by user-supplied error bounds. Here we discuss in detail the error control scheme for significant common tree generation.

Our error control mechanism limits error in magnitude and phase responses. We first choose a set of *sample frequencies* $f_i, i = 1, 2, \ldots, n_f$, in the range of interest, then compute and record the numerical values of N_2 and D_2 at these frequencies. We choose a relative threshold η and generate all product terms in N_2 or D_2 that are larger in magnitude than $\eta|N_2|$ or $\eta|D_2|$ for at least one of the frequencies $f_i, i = 1, 2, \ldots, n_f$, respectively. Product terms with different powers of s are generated separately as described in Section 4 but treated uniformly. We choose an error bound ε_{magn} for the magnitude and another error bound ε_{phas} for the phase for the simplified expression. If the generated product terms constitute a symbolic expression whose magnitude and phase errors with respect to T_2 at every frequency $f_i, i = 1, 2, \ldots, n_f$, are smaller than ε_{magn} and ε_{phas}, respectively, our procedure terminates. Otherwise we repeatedly reduce η and generate more product terms for both N_2 and D_2. A minimum value of η, η_{min}, is specified and η is reduced until either the symbolic expression is accurate enough or η_{min} is reached, when the procedure terminates.

Error_Control(two-graphs for N_2 and D_2, ε_{magn}, ε_{phas}, η_{max}, η_{min}, ξ , sample frequencies $f_i, i = 1, 2, \ldots, n_f$)

1. **comment**: ξ is the factor used to reduce the relative threshold η in every iteration;
2. compute numerical values of N_2 and D_2 at frequencies $f_i, i = 1, 2, \ldots, n_f$;
3. compute magnitude and phase of T_2 at frequencies $f_i, i = 1, 2, \ldots, n_f$;
4. $\eta \leftarrow \eta_{max}$;
5. **do**
6. generate all product terms in N_2 or D_2 that are larger in magnitude than ηN_2 or ηD_2 for at least one frequency $f_i, i = 1, 2, \ldots, n_f$;
7. update T_{appr};
8. **comment**: T_{appr} is the simplified symbolic expression that consists of all product terms already generated;
9. compute magnitude and phase of T_{appr} at frequencies $f_i, i = 1, 2, \ldots, n_f$;
10. **if** magnitude error and phase error of T_{appr} with respect to T_2 are smaller than ε_{magn} and ε_{phas} for all $f_i, i = 1, 2, \ldots, n_f$ **then**
11. **output** T_{appr}; **return** (ok);
12. **endif**
13. $\eta \leftarrow \eta \cdot \xi$;
14. **while** $\eta \geq \eta_{min}$;
15. **return** (warning).

Our error control scheme is different from the one in [22] and [20], where individual powers of s in both numerator and denominator are approximated, in that product terms with different powers of s are treated uniformly according to their relative magnitude with respect to the numerator or denominator rather than individual powers of s.

Specified matching conditions and mismatch variables for small signal model elements are incorporated after the symbolic expression is generated, allowing the effect of element mismatch to be readily observed. Product terms are collected after matching and the size of the expression reduced as part of an *algebraic post-processing* step.

7. Implementation and Results

The above unified approach has been implemented in a symbolic analysis program. For integrated circuits, DC analysis is carried out with SPICE and the program reads the small signal model element values for semiconductor devices from the SPICE output. Complete small signal models for bipolar and MOS transistors are used. The program is able to generate any network function in the complex frequency domain.

The first circuit example analyzed is the bipolar opamp $\mu A741$ containing 26 transistors and 11 resistors. An approximate symbolic expression for the voltage gain, valid up to the unity-gain frequency, containing a total of 57 product terms before matching and algebraic post-processing was obtained in 18.5 seconds on a Sun SPARCStation2. The frequency response of the approximate symbolic expression is compared to the numerical simulation results using SPICE in Fig. 1. The statistics of the symbolic expression are given in Table 1.

The second circuit example is a CMOS cascode opamp containing 22 transistors given in Fig. 2. An approximate symbolic

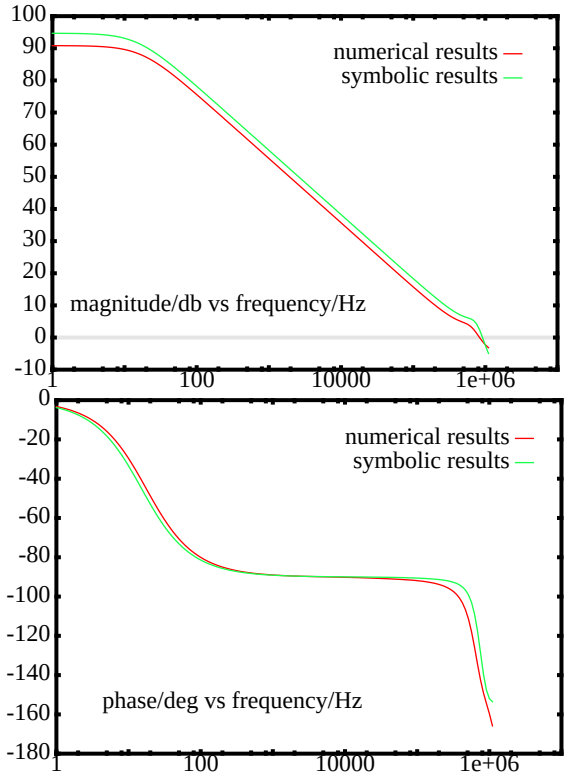

Figure 1. Voltage gain of $\mu A741$ as given by the approximate symbolic expression.

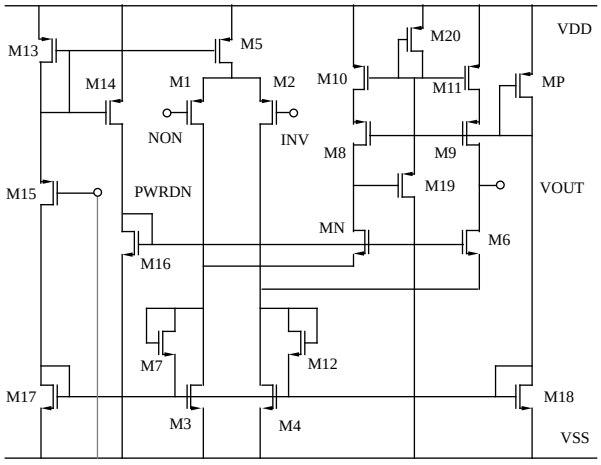

Figure 2. CMOS cascode opamp.

expression for the voltage gain, valid up to the unity-gain frequency, containing a total of 245 product terms before matching and algebraic post-processing was obtained in 20.4 seconds on the same machine. The frequency response of the approximate symbolic expression is compared to the numerical simulation results using SPICE in Fig. 3. Also shown is the frequency response of voltage gain approximated by different numbers of product terms in the expression. The statistics of the symbolic expression are given in Table 1.

The third circuit example is a rail-to-rail opamp [24] containing 39 transistors. An approximate symbolic expression for the voltage gain, again valid up to the unity-gain frequency, contain-

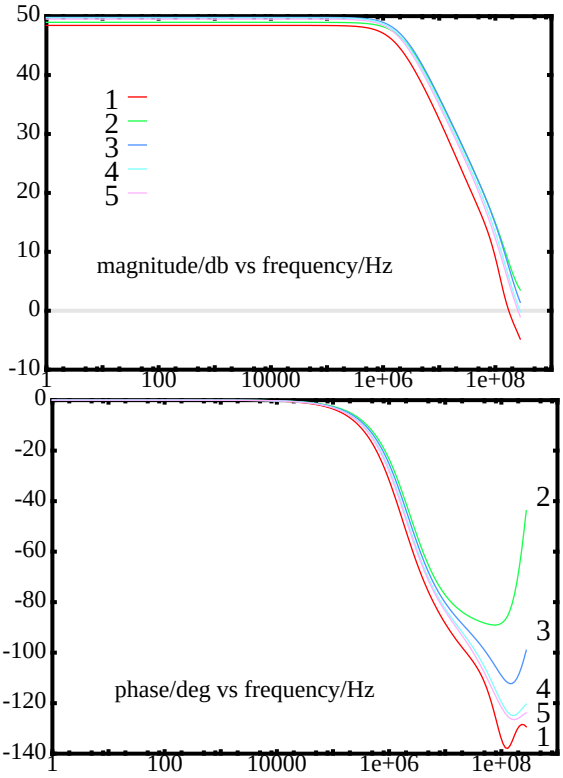

Figure 3. Voltage gain of CMOS cascode opamp as approximated by different number of product terms in the symbolic expression: 1) numerical results; 2) 54 terms; 3) 99 terms; 4) 245 terms; 5) 450 terms.

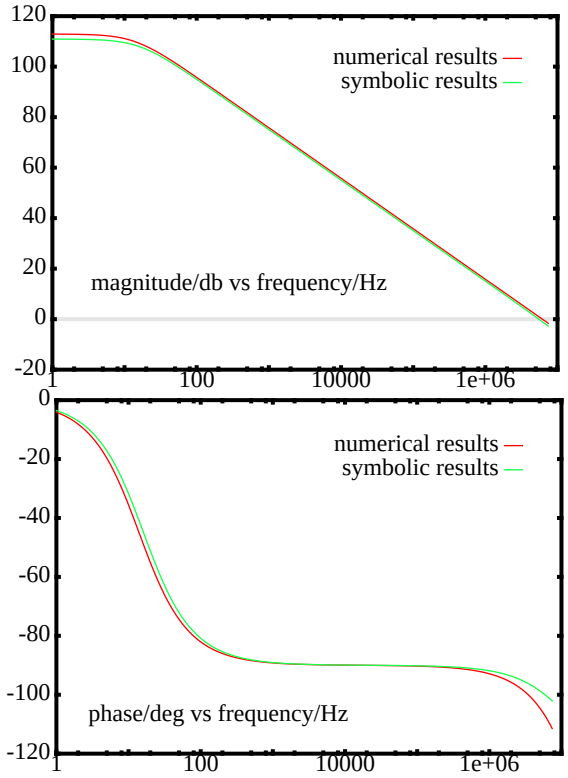

Figure 4. Voltage gain of rail-to-rail opamp as given by the approximate symbolic expression.

ing a total of 36 product terms before matching and algebraic post-processing was obtained in 63.4 seconds. The frequency response of the approximate symbolic expression is compared to the numerical simulation results using SPICE in Fig. 4. The statistics of the symbolic expression are given in Table 1.

The last circuit example is the bipolar opamp $\mu A725$ with 26 transistors and 20 resistors. An approximate symbolic expression for the differential mode input impedance containing a total of 39 product terms before matching and algebraic post-processing was obtained in 41.6 seconds. Again, the frequency response of the approximate symbolic expression is compared to the numerical simulation results using SPICE in Fig. 5. The statistics of the symbolic expression are given in Table 1.

Table 1 contains upper and lower bounds on the numbers of product terms in the original unsimplified expressions of the numerator and denominator. They are computed as

$$\min\{\det(A_v A_v^t), \det(A_i A_i^t)\} \text{ and } \left|\det(A_i A_v^t)\right|, \qquad (25)$$

respectively, where A_v and A_i are the reduced incidence matrices of the *v*-graph and *i*-graph and

$$\det(A_v A_v^t),\ \det(A_i A_i^t) \text{ and } \left|\det(A_i A_v^t)\right| \qquad (26)$$

are the number of *v*-graph trees, the number of *i*-graph trees and the difference in the numbers of common trees with plus sign and minus sign, respectively [17].

In all examples, the number of circuit elements and the highest powers of s in the symbolic expression are reduced during the approximation. Numerical accuracy is achieved with reasonable numbers of product terms before matching and post-processing compared to astronomical numbers of product terms in

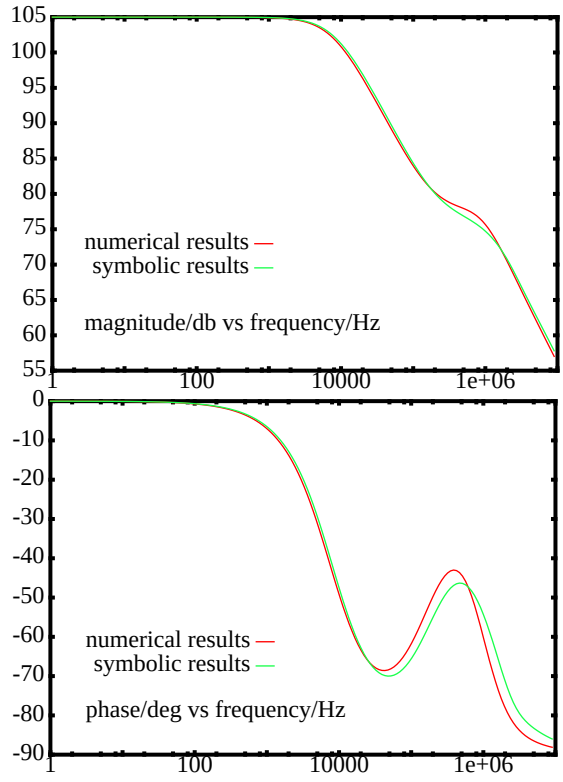

Figure 5. Differential mode input impedance of $\mu A725$ opamp as given by the approximate symbolic expression.

Table 1. Statistics for approximate symbolic expressions for voltage gain or differential mode input impedance

Circuit name	741		Cascode		Rail-to-rail		725	
	Nu.	De.	Nu.	De.	Nu.	De.	Nu.	De.
Original # g elements	62	62	47	47	79	81	82	82
# g elements after simplifying individual two-graphs	18	24	26	29	45	50	82	82
# g elements after simplifying both two-graphs together	7	14	11	14	19	24	9	9
Original # C elements	39	39	40	42	58	58	46	46
# C elements after simplifying individual two-graphs	19	21	25	28	41	41	46	46
# C elements after simplifying both two-graphs together	7	9	6	9	9	9	5	5
Original highest power of s	22	22	12	12	24	24	31	32
Highest power of s after simplifying individual two-graphs	14	16	12	12	24	24	31	32
Highest power of s after simplifying both two-graphs together	6	8	6	7	8	8	5	5
Highest power of s in the final expression	4	4	4	6	2	7	3	4
Upper bound on # product terms in the original expression	1.4e17	1.4e17	2.0e11	4.3e11	8.9e20	6.9e22	2.6e22	6.9e22
Lower bound on # product terms in the original expression	1.1e14	3.3e16	4.4e9	1.2e11	0	1.2e22	5.8e21	1.2e22
# product terms in the approximate symbolic expression before matching and algebraic post-processing	20	37	57	188	11	25	20	19
# elements in a product term	6	8	7	7	8	8	6	7
# v-graph spanning trees generated	64	182	179	454	117	3006	66	110

the original unsimplified expressions.

8. Conclusion

We have presented a new unified approach to the approximate symbolic analysis of large analog circuits. In addition to being able to handle much larger analog circuits than previously analyzed, our approach possesses the following virtues:

- The simplified symbolic expression is in expanded format. It is *irreducible*, or *cancellation-free*. The expanded format facilitates post-processing of the expression for different applications.
- Every product term in the simplified expression, before matching and algebraic post-processing, corresponds to a unique product term in the unsimplified irreducible sum-of-products expression for both the numerator and the denominator. No *alien* product terms are introduced.
- The error in the *contribution* of every remaining circuit element to both the numerator and denominator is limited in the sensitivity-based two-graph simplification. This offers good protection of the information content of the symbolic expression against distortion during approximation.
- Generation of significant common trees provides maximum flexibility to any error control mechanism for symbolic expressions in expanded format.

References

[1] M. Amadori, R. Guerrieri and E. Malavasi, "Symbolic Analysis of Simplified Transfer Functions," *Analog Integ. Circuits Sig. Processing*, Vol. 3, No. 1, pp. 9-29, 1993.

[2] S.-M. Chang, J. F. MacKay and G. M. Wierzba, "Matrix reduction and numerical approximation during computation techniques for symbolic analog circuit analysis," *Proc. IEEE ISCAS*, 1992, pp. 1153-1156.

[3] C. S. Chiang, A Perturbation Approach to the Symbolic Analysis of Analog Circuits, Ph.D. Dissertation, Yale University, New Haven, 1992.

[4] L. O. Chua and P. M. Lin, Computer-aided Analysis of Electronic Circuits: Algorithms and Techniques, Englewood Cliffs, New Jersey: Prentice-Hall, 1975.

[5] T. H. Cormen, C. E. Leiserson and R. L. Rivest, Introduction to Algorithms, Cambridge, Massachusetts: The MIT Press, 1990.

[6] F. V. Fernandez, A. Rodriguez-Vazquez and J. L. Huertas, "Interactive AC modeling and characterization of analog circuits via symbolic analysis," *Analog Integ. Circuits Sig. Processing*, Vol. 1, No. 3, pp. 183-208, 1991.

[7] F. V. Fernandez, A. Rodriguez-Vazquez, J. D. Martin and J. L. Huertas, "Formula Approximation for Flat and Hierarchical Symbolic Analysis," *Analog Integ. Circuits Sig. Processing*, Vol. 3, No. 1, pp. 43-58, 1993.

[8] H. N. Gabow, "Two algorithms for generating weighted spanning trees in order," *SIAM J. Computing*, Vol. 6, pp. 139-150, Mar. 1977.

[9] G. Gielen, H. Walscharts, W. Sansen, "ISAAC: a symbolic simulator for analog integrated circuits," *IEEE J. Solid-State Circuits*, Vol. SC-24, pp. 1587-1597, Dec. 1989.

[10] G. Gielen and W. Sansen, Symbolic Analysis for Automated Design of Analog Integrated Circuits, Boston: Kluwer Academic Publishers, 1991.

[11] G. Gielen, P. Wambacq and W. Sansen, "Symbolic analysis methods and applications for analog circuits: a tutorial overview," *Proc. IEEE*, Vol. 82, pp. 287-304, Feb. 1994.

[12] M. M. Hassoun and P. M. Lin, "A new network approach to symbolic simulation of large-scale networks," *Proc. IEEE ISCAS*, 1989, pp. 806-809.

[13] M. M. Hassoun and K. S. McCarville, "Symbolic Analysis of large-scale networks using a hierachical signal flowgraph approach," *Analog Integ. Circuits Sig. Processing*, Vol. 3, No. 1, pp. 31-42, 1993.

[14] J.-J. Hsu and C. Sechen, "Low-frequency symbolic analysis of large analog integrated circuits," *Proc. Custom Integ. Circuits Conf.*, 14.7.1-5, May 1993.

[15] J.-J. Hsu and C. Sechen, "Fully symbolic analysis of large analog integrated circuits," *Proc. Custom Integ. Circuits Conf.*, pp. 457-460, May 1994.

[16] A. Liberatore and S. Manetti, "SAPEC - a personal computer program for the symbolic analysis of electric circuits," *Proc. IEEE ISCAS*, 1988, pp. 897-900.

[17] P. M. Lin, Symbolic Network Analysis, *Studies in Electrical and Electronic Engineering 41*, New York: Elsevier, 1991.

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

[19] S. J. Seda, M. G. R. Degrauwe and W. Fichtner, "A symbolic analysis tool for analog circuit design automation," *IEEE/ACM ICCAD, Digest Tech. Papers*, 1988, pp. 488-491.

[20] S. J. Seda, M. G. R. Degrauwe and W. Fichtner, "Lazy-expansion symbolic expression approximation in SYNAP," *IEEE/ACM ICCAD, Digest Tech. Papers*, 1992, pp. 310-317.

[21] J. A. Starzyk and A. Konczykowska, "Flowgraph Analysis of Large Electronic Networks," *IEEE Trans. Circuits and Syst.*, Vol. CAS-33, pp. 302-315, Mar. 1986.

[22] P. Wambacq, G. Gielen and W. Sansen, "A cancellation-free algorithm for the symbolic simulation of large analog circuits," *Proc. IEEE ISCAS*, 1992, pp. 1157-1160.

[23] P. Wambacq, F. V. Fernandez, G. Gielen and W. Sansen, "Efficient symbolic computation of approximated small-signal characteristics," *Proc. Custom Integ. Circuits Conf.*, pp. 461-464, May 1994.

[24] W.-C. S. Wu et al, "Digital-compatible high-performance operational amplifier with rail-to-rail input and output ranges," *IEEE J. Solid-State Circuits*, Vol. SC-29, pp. 63-66, Jan. 1994.

[25] Q. Yu and C. Sechen, "Generation of color-constrained spanning trees with application in symbolic circuit analysis," *Proc. 4th Great Lakes Symp. VLSI*, pp. 252-255, Mar. 1994.