Symbolic Analysis of Linear Circuits with the Determinant Tree Diagram

Roman Dmytryshyn* and Benedykt Rodanski**

Abstract – **In this paper we propose a new approach to solving a problem of the large number of arithmetical operations in generation of formulae for symbolic analysis of linear circuits. This new approach – called the Determinant Tree Diagram (DTD) method – is based on generating permutations and is an alternative to the well known Determinant Decision Diagram (DDD) method. DTD can be applied in practice to generation of the symbolic solution of determinant of a sparse matrix (e.g. the MNA matrix) in either the bracket notation or the Reverse Polish Notation.**

1 INTRODUCTION

The main goal of symbolic circuit analysis is the development of methods and algorithms for generation of formulae, useful in analysis and design of electronic circuits and systems. It does not matter what methods or tools are used to achieve this goal (as long as the time needed to obtain a formula is acceptable); what matters is the accuracy and execution speed of the obtained formula on a given arithmetic processor. The formulae may be obtained manually or automatically, using commercially available software or specialised, non-commercial packages. Each method has its advantages and disadvantages. Until today there is no single universal method of formula generation which would not have some disadvantages.

This paper is concerned with generation of matrix determinant, where matrix elements are unique symbols. A very common approach to generating symbolic transfer functions of electronic circuits is based on solving circuit equations using Cramer 's rule [1], which requires calculating determinants of symbolic matrices. The scope of this paper is limited to generating determinant formulae without fractions. Such formulae, as opposed to topological formulae, contain subtraction operations even for passive circuits. One advantage of our approach is that, in some cases, generated formulae have smaller number of arithmetic operations compared to formulae generated using topological methods [1]. To avoid possible loss of accuracy resulting from subtractions, accumulation of positive and negative terms can be done separately and one subtraction operation is then performed at the end.

The method presented is, from the point of view of the final result, very similar to the DDD method [2]. We, however, are using completely different determinant tree, with special properties that allow generation of symbolic determinants in both classical (with brackets) and the Reverse Polish Notation (RPN) forms.

2 PROPERTIES OF THE DETERMINANT TREE (DT)

Every tree consists of vertices and branches. Let us characterise the properties of our DT.

DT has four types of vertices: root, A- and B-type vertex, and leaf. Their graphical representations are shown in Fig. 1. The B-type vertex (B-vertex) differs from the A-vertex by having several outgoing branches on its right hand side. Every vertex, except the root, can be white or black. The root is always white.

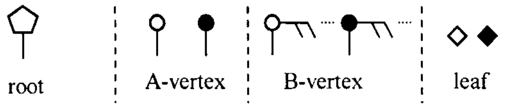


Figure 1: The types of DT vertices

The DT of an $n \times n$ matrix has three main properties:

- the maximum number of paths from the root to the leafs is equal to the number of permutations of n matrix elements where no two elements can belong to the same row or column,
- there is always an even number of black elements in a path,
- a leaf is only connected to an A-vertex.

Labelling of DT branches depends on the required notation of the formula. When generating a RPN formula, to the left of every branch we put the corresponding matrix element and to the right, except a leaf branch, the multiplication symbol (Fig. 2a, b). For branches originating from the right of a B-vertex we put a plus sign above the multiplication

[*] Rzeszów University of Technology, Department of Electrical and Computer Engineering, 2 W. Pola Str., 35-959 Rzeszów, Poland. E-mail: rdmytr@prz.rzeszow.pl

[**] University of Technology, Sydney (UTS), Faculty of Engineering, P.O. Box 123, Broadway, NSW 2007, Australia. E-mail: ben.rodanski@uts.edu.au

symbol if the bottom vertex is white; a minus sign otherwise.

While generating a classical formula, labelling of branches differs in putting the matrix elements, multiplication symbols and appropriate signs to the left of branches (Fig. 2c, d). In addition, we put brackets around the B-vertices.

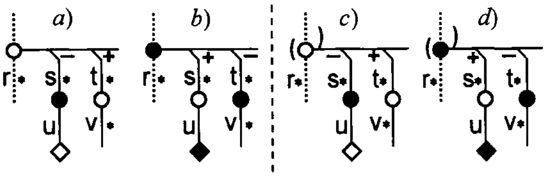


Figure 2: The rules of coding of the DT-branches for generation of symbolic determinant for:
a,b – Reverse Polish Notation; *c, d* – brackets notation

In order to obtain a formula we go around the DT anticlockwise, as shown in Fig. 3, and note all encountered symbols.

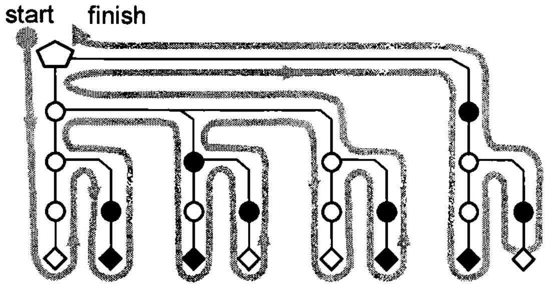


Figure 3: A trip around the DT

3 GENERATION OF SYMBOLIC DETERMINANT

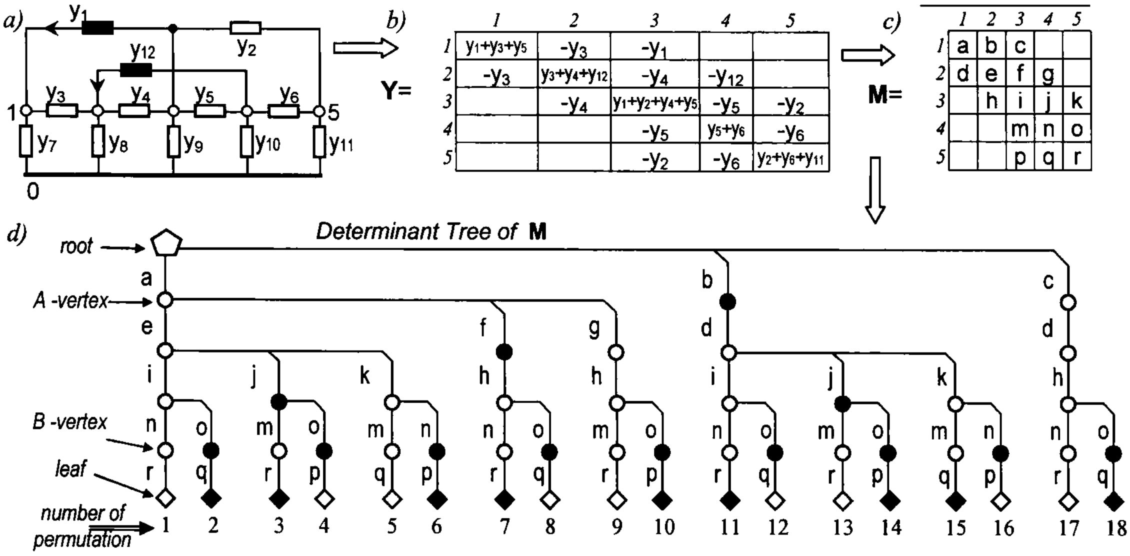
To illustrate the algorithm for generation of a symbolic determinant we'll use a circuit shown in Fig. 4a. This circuit contains unistors, so it is modelled with controlled sources and its node admittance matrix **Y** is not symmetrical. We create matrix **M** by replacing each nonzero entry of **Y** by a unique symbol. The determinant of **M** is given by:

det**M** = **aeinr**-aei**oq**-ae**jmr**+aej**op**+ae**kmq**-aek**np**-a**fhnr**+ afh**oq** +a**ghmr** –agh**op** -**bdinr** +bdi**oq** +bd**jmr** -bdj**op** - bd**kmq** +bdk**np** +**cdhnr** –cdh**oq** 68[*], 17[±]

where the bold face letters in each term show symbols new in this term, comparing with the preceding term.

Using the above terms we can plot the determinant tree, as shown in Fig. 4d. Each term in the determinant has a corresponding path from the root to the leaf, so the number of leafs is equal to the number of terms. A leaf is black for a negative term and white for a positive term. The number of vertices in each path is equal n+1. To the left of each branch we put the corresponding symbol from **M**. We note that in the determinant tree only new symbols (bold in the formula above) are used.

Since the number of black vertices in each path must be even, we may need to change the color of one of the vertices in some paths. Going down from the root, the first white vertex that does not belong to the previous path changes color to black.



Y=

	1	2	3	4	5
1	$y_1+y_3+y_5$	$-y_3$	$-y_1$		
2	$-y_3$	$y_3+y_4+y_{12}$	$-y_4$	$-y_{12}$	
3		$-y_4$	$y_1+y_2+y_4+y_5$	$-y_5$	$-y_2$
4			$-y_5$	y_5+y_6	$-y_6$
5			$-y_2$	$-y_6$	$y_2+y_6+y_{11}$

M=

	1	2	3	4	5
1	a	b	c		
2	d	e	f	g	
3		h	i	j	k
4			m	n	o
5			p	q	r

Figure 4: An example: *a*- circuit, *b*- **Y**-matrix, *c*- **M**-matrix, *d*- determinant tree

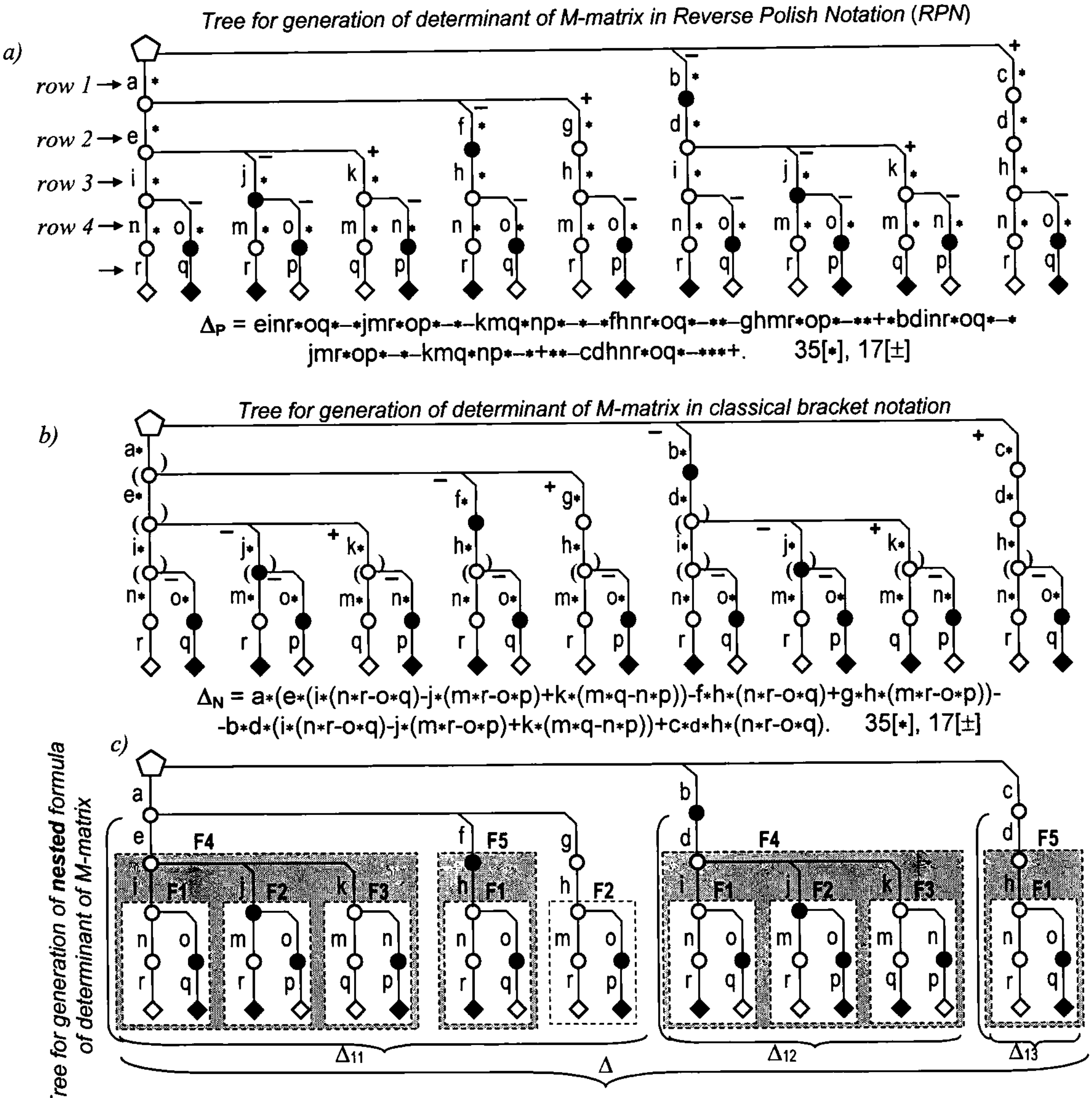

Tree for generation of determinant of M-matrix in Reverse Polish Notation (RPN)
a)
row 1→a
row 2→e
row 3→i
row 4→n
ΔP = einr*oq*–*jmr*op*–*–kmq*np*–*–*fhnr*oq*–**–ghmr*op*–**+*bdinr*oq*–*
jmr*op*–*–kmq*np*–*+**–cdhnr*oq*–***+. 35[*], 17[±]
Tree for generation of determinant of M-matrix in classical bracket notation
b)
ΔN = a*(e*(i*(n*r-o*q)-j*(m*r-o*p)+k*(m*q-n*p))-f*h*(n*r-o*q)+g*h*(m*r-o*p))-
-b*d*(i*(n*r-o*q)-j*(m*r-o*p)+k*(m*q-n*p))+c*d*h*(n*r-o*q). 35[*], 17[±]
c)
Tree for generation of nested formula of determinant of M-matrix
F4
F5
F1
F2
F3
Δ11
Δ12
Δ13
Δ

Figure 5: An example of generation of the formulae of determinant of matrix **M** based on DTD

3.1 Generation of flat formulae

Figure 5a shows a tree for generation of flat determinant formula in the RPN. Branches are labeled according to Fig. 2a,b. Traveling around this tree, according to Fig. 3, and writing down all encountered symbols results in the RPN formula, shown below the tree (Δ_P = einr*oq*...).

If the branches of the DT are labeled according to Fig. 2c,d, traveling around the tree and writing down all encountered symbols results in a classical formula with brackets (Δ_N = a*(e*(i*(n*... in Fig. 5b).

Both flat formulae, RPN and bracketed (the last one is also called *nested*), have considerably smaller number of multiplications than the expanded formula.

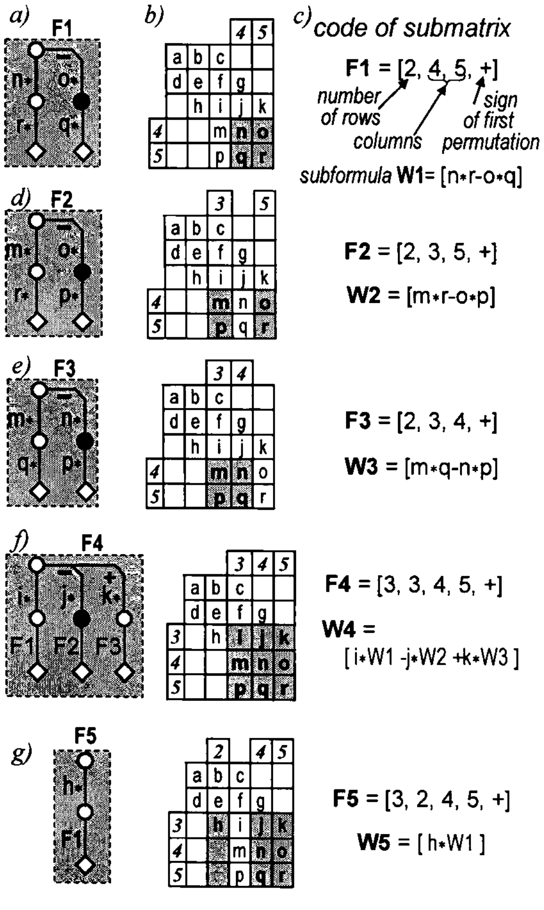
3.2 Generation of sequential (hierarchical) formulae

A closer inspection of the bracketed formula shows that several sub-expressions appear more than once in the formula. A considerable amount of execution time can be saved if these sub-expressions are calculated only once and replaced by a unique symbol, as shown in Fig. 5c. This results in a hierarchical formula, also called the sequence of expressions (SoE). The number of arithmetic operations is further reduced in the SoE form.

3.2.1 Coding of the cofactors

Consider again the tree in Fig. 5c with all relevant sub-trees marked. Those sub-trees represent several cofactors which may appear many times in both the numerator and the denominator of a network function (transmittance). It will therefore be very advantageous to extract those sub-expressions and replace them by unique symbols to be used in the final formula.

To illustrate this procedure, let us draw separately all marked sub-trees F1 – F5 from Fig. 5c. This is shown in Fig. 5a,d,e,f,g. Each of these sub-trees represents a certain minor of **M** (Fig. 6b). In order to facilitate computer generation of expressions, we'll code each sub-tree using a variable length array, as shown in Fig. 6c. The first entry in the array is the number (k) of last consecutive rows of the corresponding minor. Next k entries are the column indices. The last entry shows the sign of the first permutation, obtained from the Cartesian product algorithm applied to sets of column indices in table K (Fig. 4c). (This sign can be calculated each time the formula is generated, but it is advantageous to obtain the sign only once; it saves time later, when the final SoE is generated.) Each array F is used to obtain a symbolic cofactor W.



c) code of submatrix

F1 = [2, 4, 5, +]

number of rows; columns; sign of first permutation

subformula W1= [n*r-o*q]

F2 = [2, 3, 5, +]

W2 = [m*r-o*p]

F3 = [2, 3, 4, +]

W3 = [m*q-n*p]

F4 = [3, 3, 4, 5, +]

W4 = [i*W1 -j*W2 +k*W3]

F5 = [3, 2, 4, 5, +]

W5 = [h*W1]

Figure 6: The examples of coding of sub-tree, sub-matrix and sub-formula

3.2.2 Hierarchical determinant tree

Figure 7 shows the hierarchical structure of the DT for SoE generation. The new tree has less paths and the paths are of variable length. It is important to note that the properties of the hierarchical DT (HDT) are identical with the original DT.

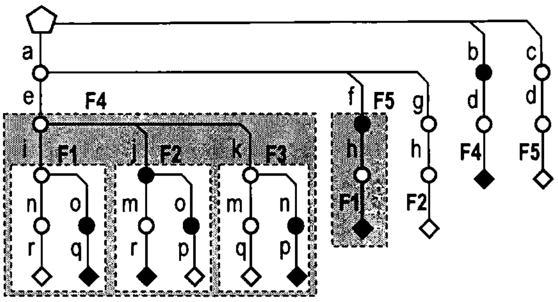
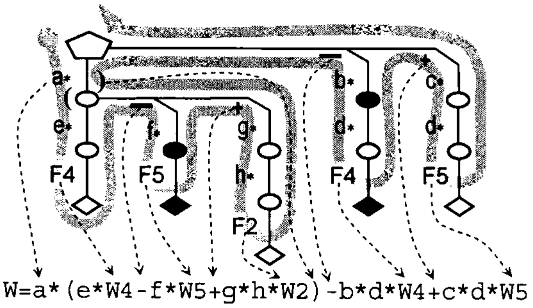


Figure 7: The hierarchical structure of DT

3.2.3 Symbolic determinant in the SoE form

Without loss of generality we can consider the classical (bracketed) formula only. To obtain the determinant formula in the sequence of expressions form we first generate all cofactors (starting from the deepest level) and then replace all sub-trees in the original DT by the symbols representing the corresponding minors. The resulting HDT for our example is shown in Fig. 8.



```
W=a*(e*W4-f*W5+g*h*W2)-b*d*W4+c*d*W5
```

Figure 8: The hierarchical DT (HDT) for bracketed symbolic determinant

Note that the number of paths in the HDT in Fig. 7 is 3.7 times smaller than the number of paths in the original DT from Fig. 4d. Traveling around the HDT and writing down all encountered symbols we obtain the final formula in the SoE form:

```
W1=n*r-o*q; W2=m*r-o*p; W3=m*q-n*p;
W4=i*W1-j*W2+k*W3; W5=h*W1;
W=a*(e*W4-f*W5+q*h*W2)-b*d*W4+c*d*W5
                                [*]=19, [±]=9.
```

Counting the number of arithmetic operations in the SoE shows a considerable improvement over the flat formulae (both expanded and nested): the number of multiplications decreased 68/19 = 3.6 and 35/19 = 1.8 times, the number of additions/ subtractions decreased 17/9 = 1.9 times.

4 CONCLUSIONS

A new method for generation of symbolic determinant formulae has been presented. The method is especially effective for determnants of sparse matrices. It can generate both flat and hierarchical formule in either classical bracketed (nested) or the RPN forms. Since all network functions can be expressed as ratios of certain determinants (of usually very sparse matrices), our technique has immediate applications in symbolic circuit analysis.

From the point of view of computer algebra, generation of symbolic determinant formulae in bracketed (nested) form is a well known problem of formula factorization. An attempt to address this problem has been made in commercial packages as MAPLE or MATHEMATICA. Unfortunately, factorization belongs to the so called NP-hard combinatorial problems, and the solution obtained from commercial packages are not always satisfactory. The DTD algorithm, presented in this paper, may be used to improve the efficiency of the factorization procedures, and is not limited to symbolic circuit analysis.

The DTD has excellent educational values, based on practical application of important elements of graph theory and combinatorics. In comparison with symbolic LU-decomposition, as in SCAPP [6], or symbolic Gaussian elimination, as in STAINS [7], the DTD evaluation is division-free. Early experimental results indicate that DTD performs at least as good as the DDD method [2, 3]. The DTD method is conceptually simpler and useful for both the education process ("Discrete Mathematics" [5], "Circuit Theory," etc.) and implementation in CAD of electronic circuits and systems.

References

[1] P.-M. Lin, *Symbolic Network Analysis*, Amsterdam: Elsevier, 1991.

[2] C.-J. Shi, X. Tan, "Canonical Symbolic Analysis of Large Analog Circuits with Determinant Decision Diagrams," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 19, no. 1, 2000, pp. 1-18.

[3] W. Verhaegen, G. Gielen, "Symbolic Determinant Decision Diagrams and Their Use for Symbolic Modeling of Linear Analog Integrated Circuits," *Analog Integrated Citcuits and Signal Processing*, 31, Kluwer Academic Publishers, 2002, pp. 119-130.

[4] R. Dmytryshyn, A. Kubaszek, "Multimethodical Approach and Sequence of Expressions Generation for Acceleration of Repetitive Analysis of Analog Circuits," *Analog Integrated Circuits and Signal Processing*, 31, Kluwer Academic Publishers, 2002, pp. 147-159.

[5] R. Dmytryszyn, G. Dralus, *Discrete Mathematics (algorithms, exercises, project)*, Rzeszów University of Technology Publishing, Rzeszów, Poland, 2003. p.125 (in Polish).

[6] M.M. Hassoun, P.-M. 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. 4, pp. 201-211, April 1995.

[7] M. Pierzchała, B. Rodanski, "Generation of sequential symbolic network functions for large-scale networks by circuit reduction to a two-port," *IEEE Trans. on Circuits and Systems - I: Fundamental Theory and Applications*, vol. 48, no. 7, pp. 906-909, July 2001.