COMPUTER IMPLEMENTATION OF TWO TOPOLOGICAL UNIQUENESS TESTS

by

T. S. Parker, M. P. Kennedy, Y. Liao and L. O. Chua

Memorandum No. UCB/ERL M85/78

1 October 1985

COMPUTER IMPLEMENTATON OF TWO TOPOLOGICAL UNIQUENESS TESTS

by

T. S. Parker, M. P. Kennedy, Y. Liao and L. O. Chua

Memorandum No. UCB/ERL M85/78

1 October 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

title 19

Computer Implementation of Two Topological Uniqueness Tests

Thomas S. Parker, Michael Peter Kennedy,
Youlin Liao and Leon O. Chua

Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley

ABSTRACT

This paper presents several new algorithms which are used to implement two recently published uniqueness theorems applicable to nonlinear resistive circuits containing independent sources, two-terminal resistors and linear controlled sources. The algorithms and the two programs which use them are described and results are presented for some examples from the original papers.

1. Introduction

Recently, several topological tests have been developed to determine whether a nonlinear resistive circuit (of a specific type) possesses a unique solution [1, 2, 3, 4]. These tests are unusual from a circuit-analysis point of view because the uniqueness criteria are couched in purely topological terms rendering standard circuit-analysis algorithms ineffective.

For example, the main theorem of [1] requires the generation of a large number of subcircuits, each of which is obtained from the circuit under analysis by open- or short-circuiting each resistor and by zeroing or leaving intact each controlled source. If n_r and n_{cs} are the number of resistors and controlled sources in the original circuit, then there are $2^{(n_r + n_{cs})}$ different subcircuits. Except in the case of a very small circuit, the generation of these subcircuits is tedious and virtually impossible to perform reliably by hand.

This paper presents several new algorithms which can be used to write efficient programs which implement the topological uniqueness tests. To demonstate the algorithms, we also report on two computer programs— UNIQ which implements Theorem 8 of [1] and UNIQF which implements Theorem 2 of [2].

Theorem 8 (see Appendix 1 for a precise statement of the theorem) is a general method for determining uniqueness and existence of solution for circuits containing any number of independent voltage and current sources, any number of linear controlled sources (of all four types) and any number of strictly increasing, two-terminal resistors. The method yields a yes/no answer indicating whether, for each set of allowed circuit parameters, the circuit possesses a unique solution. The method is purely topological in nature and requires no floating-point calculations.

Research supported in part by the Hertz Foundation, by the Office of Naval Research under Contract N00014-76-C-0572 and by National Science Foundation Grant ECS8542885.

Theorem 2 (see Appendix 2) pertains to circuits with any number of independent sources, any number of linear CCCSs or linear VCVSs (but not both) with finite controlling coefficients and any number of strictly increasing, two-terminal resistors. Like Theorem 8, Theorem 2 results in a yes/no answer indicating whether, for each set of allowed circuit parameters, the circuit possesses a unique solution. However, unlike Theorem 8, Theorem 2 does require some floating-point calculation since it deals with the bounds of the controlling coefficients.

It is important to realize exactly what information these two theorems provide. A positive result means that, for each set of allowed circuit parameters, there is one and only one solution of the circuit; the theorems yield no information as to what that unique solution is, just that it exists. Different circuit parameters may (and usually do) result in different solutions. A negative result means that, for some set of allowed circuit parameters, there is either no solution or more than one solution to the circuit; however, there may exist different sets of allowed circuit parameters (perhaps all other sets) for which the circuit does possess a unique solution.

Section 2 describes UNIQ using examples drawn from the original paper and Section 3 similarly describes UNIQF. Section 4 delves into the details of the algorithms used by UNIQ. Instead of describing UNIQF in complete detail, in Section 5 we examine only the differences between the two programs. Section 6 supplies supplemental details about the programming language used, on which machines the programs run and how to obtain copies of the programs.

2. Overview and Examples of UNIQ

UNIQ reads its input from a user-specified SPICE-like data file. The first line of the input file is taken as the title of the circuit. Each subsequent line describes a circuit element by using three fields:

name topology value.

The *name* field consists of a string of alphanumeric characters, the first of which identifies the element type. Theorem 8 deals with just seven different types of circuit elements, identified by names beginning with R (resistor), I (independent current source), V (independent voltage source), E (VCVS), F (CCCS), G (VCCS) and H (CCVS).

The *topology* field consists of either one (for resistors and independent sources) or two (for controlled sources) pairs of integers, denoting the terminal node numbers of the corresponding one- or two-port. For a controlled source, the first pair of integers denotes the output port and the second denotes the input (controlling) port.

The *value* field characterizes the constitutive relation of the circuit element. Element values are not used by UNIQ (since Theorem 8 is topological in nature), so the *value* field need not be entered.

For example,

```
R  2  3
```

represents a resistor connected between nodes 2 and 3 of the circuit, while

```
F  1  3  4  2
```

is a current-controlled current source with output port connected between nodes 1 and 3 and controlling port connected between nodes 4 and 2.

The output of UNIQ comes in two forms, *normal* and *verbose*, either of which may be selected by the user. In both modes, UNIQ prints the title of the circuit and a copy of the names and terminal connections of each element in the circuit together with a list of the loops and cutsets, if any, which violate the Interconnection Assumption. UNIQ then generates controlled-source graphs and tests for complementary tree structure. In verbose mode, for *every* controlled-source graph with complementary tree structure, UNIQ prints the state of each resistor (open or short) and controlled source (intact or zeroed), and indicates which type of complementary tree structure (positive or negative) the graph possesses. For each intact controlled source, the terminal connections and directions (as specified by Theorem 8) of the input and output branches are also given. In normal mode (the default), this information is given only for each new complementary tree structure encountered, that is, for the first controlled-source graph found with positive complementary tree structure, if any, and the first found with negative complementary tree structure, if any. In normal mode, testing continues until UNIQ has enough information to conclude whether or not the circuit has a unique solution, at which time UNIQ prints its verdict. In verbose mode, UNIQ continues processing until is has found all controlled-source graphs with complementary tree structure.

All examples in [1] were tested on UNIQ and our results agree with [1] (except as noted below). We now present a few of the examples in [1].

Example 1 (Fig. 6(c) of [1]): Consider the circuit of Fig. 1, which does not satisfy the Interconnection Assumption. Fig. 2 shows the output of UNIQ when run in normal mode. First, UNIQ finds the illegal loop composed of the input and output branches of H1, the input branch of H2, and the independent voltage source V1. In this example, there is no positive complementary tree structure so the first negative complementary tree structure which UNIQ finds is listed and UNIQ concludes that this circuit has a unique solution.

Example 2 (Fig. 6(d) of [1]): Consider the circuit of Fig. 3(a), which again does not satisfy the Interconnection Assumption. UNIQ finds the illegal cutset composed of the input and output branches of G1, the input branch of G2, and the independent current sources I1 and I2. Running UNIQ in verbose mode, we find that the only valid complementary tree structure graphs are those shown in Figs. 3(b) and 3(c) (reconstructed from data in the output file (Fig. 4(a))), both of which possess negative complementary tree structure. Hence, UNIQ concludes that this circuit has a unique solution[1].

Example 3: However, if the polarity of the input port of G2 is reversed then both positive and negative complementary tree structures can be found (from output file (Fig. 4(b))), and UNIQ concludes that the solution is NOT unique.

3. Overview and Examples of UNIQF

The input format of UNIQF is identical to that of UNIQ with two exceptions. CCVSs and VCCSs are not allowed and the *value* field for CCCSs and VCVSs must contain a positive real number indicating the maximum allowed controlling coefficient.

The output format is also very similar to that of UNIQ. The main difference is that when UNIQF prints out the state of the circuit elements for a controlled-source graph, it also prints out the value returned by the determinant test. This feature is useful for finding bounds on the controlling coefficients.

[1] This result differs from that given in [1] where one of the complementary tree structures was accidentally thought to be positive.

All examples in [2] were tested using UNIQF and the results from UNIQF agree with those of [2]. Here we list just a few examples from [2] to give the reader an impression of how UNIQF works.

Example 4 (Example 5 of [2]): This is a flip-flop circuit (Fig. 5) where the four CCCSs and the nonlinear resistors represent the Ebers-Moll model of two transistors. Nishi and Chua show in [2] that if the sum of the controlling coefficients of F1 and F2, $\alpha_1 + \alpha_2 > 1.0$, the circuit will not have a unique solution. Thus it can perform as a flip-flop. Figs. 6(a) and 6(b) present the output of UNIQF for the cases $\alpha_1 + \alpha_2 = 1.001$ and $\alpha_1 + \alpha_2 = 0.999$, respectively. The results agree with those of Nishi and Chua.

Example 5 (Example 7 of [2]): This is a VCVS circuit containing two VCVS's (Fig. 7). Nishi & Chua stated in [2] that when the controlling coefficient α_1 of E1 satisfies $\alpha_1 < 1.0$, then the circuit will have a unique solution. The output of UNIQF for the cases $\alpha_1 = 1.001$ and $\alpha_1 = 0.999$ is presented in Figs. 8(a) and 8(b), respectively. The output agrees with the result of [2].

4. Discussion of the Algorithms Used by UNIQ

Here we present the algorithm for the outer level of UNIQ.

0) Initialize. Reset pos_cts and neg_cts flags.
1) Read/check input.
2) Check Interconnection Assumption. If independent sources violate it, exit with a negative result.
3) Zero independent sources.
4) For each combination of opened/shorted resistors:
    5) Perform controlled-source test.
6) If either pos_cts or neg_cts flag set, exit with positive result; else exit with negative result.

The controlled-source test, step 5), is as follows:

5.1) For each combination of intact/zeroed controlled sources:
    5.2) Determine if complementary tree structure exists; if so, determine its sign (positive or negative).
    5.3) If no complementary tree structure exists or if one with this sign has been seen before, go to 5.1).
    5.4) If Interconnection Assumption restriction violated, go to 5.1).
    5.5) If positive complementary tree structure, set pos_cts flag; else set neg_cts flag.
    5.6) If both pos_cts and neg_cts flags are set, exit with negative result.

We now discuss the details of each step.

Step 0) Initialization. Internal buffers and lists are initialized. The two result flags, pos_cts (positive complementary tree structure) and neg_cts (negative complementary tree structure) are both reset to FALSE.

Step 1) Input. As mentioned earlier, the input uses a SPICE-like format. As the input file is being read, the input lines are checked for validity. Illegal elements (capacitors, inductors, etc.) cause the program to print an "illegal element" message. If element values are supplied, they are checked for validity; any errors (negative resistances, negative controlling coefficients or nonlinear controlled sources) cause UNIQ to print an error message. This feature is

included to prevent the user from accidentally applying UNIQ to circuits which Theorem 8 is not designed to analyze. If an input error occurs, UNIQ exits only after reading the entire input file. This feature allows UNIQ to report all the input errors to the user at the same time.

Step 2) Interconnection Assumption. The Interconnection Assumption is a vital part of Theorem 8. The program must find and store all violating loops and cutsets. For details see Appendix 3.

If independent sources comprise any violating loop or cutset, the program immediately exits with a NO UNIQUE SOLUTION message.

Step 3) Zero Independent Sources. All independent sources are zeroed (operation (a) of Theorem 8).

Step 4) Open/Short Resistors. The program must open/short all possible combinations of resistors (operation (b) of Theorem 8). A recursive function is ideal for this task.

The following recursive function, *open_short(A)* performs a controlled-source test for every combination of opened and shorted resistors. *A* is the reduced node-incidence matrix.

```
begin open_short(A)
    if some resistors left in A
        store a copy of A in old_A
        open a resistor in A
        call open_short(A)
        restore A using old_A
        destroy the copy old_A
        short the same resistor in A
        call open_short(A)
    else
        perform controlled-source test on A
end open_short()
```

Note that every call to *open_short()* eliminates (opens or shorts) one resistor from the circuit. Only when there are no resistors left does *open_short()* perform a controlled-source test.

To see how *open_short()* works, call *open_short()* with a circuit containing two resistors, R1 and R2 (point A of Fig. 9). The circuit contains resistors so the **if** block is executed. A copy of the circuit is stored in memory and then R1 is opened leaving a subcircuit with just one resistor (R2) in it.

Next, *open_short()* calls itself with this subcircuit and we descend to level II of the recursion (point B). The subcircuit contains a resistor so again the **if** block is executed. After storing a copy of the circuit, R2 is opened resulting in a controlled-source graph with no resistors.

Again, *open_short()* calls itself with this controlled-source graph and we descend to level III of the recursion (point C). The controlled-source graph contains no resistors so the **else** statement is executed and a controlled-source test is performed.

Once the controlled-source test is completed, *open_short()* returns to level II. The original subcircuit is restored (point B) from the copy and the copy is then destroyed freeing memory for later use[2]. Next, R2 is shorted creating the

[2] This may seem a minor point, but in cases where the recursion goes down several levels, freeing memory is a necessity.

second controlled-source graph. *open_short()* descends to level III of the recursion by calling itself. A controlled-source test is performed and *open_short()* returns to level II. At this level the **if** block is finished and *open_short()* returns to level I. Here the original circuit is restored (point A), R1 is shorted and the whole process repeats going from A to E to F to E to G to E and then back to A.

Examination of level III of Fig. 9 shows that *open_short()* has generated all possible combinations of opened/shorted resistors for this two resistor circuit. It is clear that *open_short()* works just as well for circuits with an arbitrary number of resistors. Recursion may seem confusing at first, but the concept is fairly intuitive and the resulting code is quite compact.

Step 5) Controlled Source Test. At this point in the program, the only elements left in the circuit are controlled sources. Each controlled source must either be left intact or zeroed (operation (c') of Theorem 8).

Step 5.1) Zero/Intact Controlled Sources. UNIQ must generate all possible combinations of zeroed and non-zeroed controlled sources. A recursive function similar to *open_short()* is used. Instead of opening on the left branch and shorting on the right, this function leaves the controlled source untouched on the left branch and zeros it on the right, thereby generating all possible controlled-source graphs.

Step 5.2) Test for Complementary Tree Structure. The controlled-source graph, G_0, must be checked for complementary tree structure (see Appendix 1 for a definition).

For programming ease, we use the reduced node-incidence matrix, $A := [A_T | A_L]$ instead of the fundamental loop matrix. The following fact describes the conditions on A for G to have complementary tree structure.

Fact 1: Let G_0 have reduced node-incidence matrix $A = [A_T | A_L]$ where A_T and A_L are $n \times n$ matrices. G_0 has positive (resp. negative) complementary tree structure if and only if

$$D := (-1)^n \det A_L \det A_T$$

is positive (resp. negative).

Proof: If either A_L or A_T are singular, A does not possess complementary tree structure and $D = 0$. Suppose both A_L and A_T are nonsingular. KCL for the reduced node-incidence matrix, $Ai = 0$, implies $A_T i_T + A_L i_L = 0$ where $[i_T | i_L]$ is the partition of the branch current vector corresponding to the partition of A. Hence, $i_T = -A_T^{-1} A_L i_L$. KCL for the fundamental loop matrix, $i = B^T i_L$, implies $i_T = B_T^T i_L$. Hence, $B_T^T = -A_T^{-1} A_L$ and

$$\det B_T = (-1)^n \det(A_T^{-1} A_L) = (-1)^n \det A_L / \det A_T.$$

The sign is not changed if the division is replaced by multiplication and we have the sign of $\det B_T$ is the same as the sign of D. EOP.

Thus testing for a complementary tree structure is equivalent to finding the determinant of two square node-incidence matrices. The proof of the following well-known fact outlines an efficient algorithm for calculating the determinant of a reduced node-incidence matrix.

Fact 2: The determinant of an $n \times n$ reduced node-incidence matrix, A, is 1, 0, or -1.

Proof: By induction. If $n = 1$, the result is trivial. Suppose $n = k + 1$ and the determinant of any $k \times k$ reduced node-incidence matrix is 1, 0 or -1. Choose a column with a single non-zero entry. (If no such column exists, $\det A = 0$ since either there exists an all zero column or all columns have two entries in which case the rows sum to zero.) Using this column, expand $\det A$ by minors to get

det A = ±det M where M is the minor associated with the non-zero entry. M is a $k \times k$ matrix so det M is 1, 0 or −1. Therefore, det A is 1, 0, or −1. EOP.

This leads to a very simple and efficient algorithm for finding det A_L and det A_T:

```
begin det(A)
    if only one column in A
        return a11
    else if there is a column in A with only one non-zero entry
        call the entry aij
        create M by deleting row i, column j
        return (-1)^(i+j) det(M)
    else
        return 0
end det()
```

Once again the algorithm is recursive and the code is very compact.

Step 5.3) New Complementary Tree Structure? If no complementary tree structure exists or if a complementary tree structure does exist but possesses a sign that UNIQ has already encountered, then UNIQ need not perform any more tests on this controlled-source graph.

Step 5.4) Interconnection Assumption Restriction. The controlled-source graph is checked to see whether every loop or cutset violating the Interconnection Assumption has at least one branch still in the graph (restriction on operation (c')). If this restriction is violated, the controlled-source graph is of no interest and the next controlled-source graph should be generated.

Step 5.5) Set/Check Flags. Set either pos_cts or neg_cts to TRUE depending on the sign of the complementary tree structure. If both flags are now TRUE, exit with a NO UNIQUE SOLUTION message.

Shortcuts. A few shortcuts may be applied to eliminate unnecessary calculations.

If the circuit is not connected, no subgraph of the circuit can ever possess complementary tree structure. Hence circuits with more than one piece do not possess a unique solution. In particular, the internode voltages between nodes on different pieces are arbitrary. Similarly, if, at any level in the recursion, opening a resistor or zeroing a controlled source results in a graph with more than one component, no further recursion need be done on this particular graph.

Suppose the circuit is connected, contains no controlled sources and has no loops (resp. cutsets) of independent voltage (resp. current) sources. Since there are no controlled sources, all of the controlled-source graphs must consist solely of nodes. The only all-node graph with complementary tree structure is the graph consisting of a single node. We can obtain this single node controlled-source graph by shorting all the resistors. Hence all such circuits possess a unique solution.

If, after zeroing a controlled source, there are fewer branches than nodes, further zeroing of sources can yield at best positive complementary tree structure. If pos_cts is already TRUE (indicating that positive complementary tree structure has already been found), then further zeroing will provide no new information and recursion along this path can be terminated.

5. Discussion of Additional Algorithms used by UNIQF

Here is the algorithm for the outer level of UNIQF. For simplicity, we discuss only the CCCS case.

0') Initialize.

1') Read/check input.

2') Check Interconnection Assumption conditions 1) and 2) of Theorem 2. If the Interconnection Assumption is violated, exit with a negative result.

3') Zero independent sources.

4') For each combination of opened/shorted resistors:

5') Perform controlled-source test.

6') Exit with a positive result.

The algorithm for the controlled-source test is as follows.

5.1') For each combination of intact/zeroed CCCSs:

5.2') Determine if the branches associated with the input ports form a tree; if not, go to 5.1').

5.3') Calculate $\Delta := \det(I + CQ_L)$. If $\Delta < 0$, exit with a negative result.

All steps except step 5') are similar to UNIQ so we will only discuss step 5').

Step 5.1') Zero/Intact CCCSs. The program must generate all possible combinations of zeroed and intact CCCSs. UNIQF performs this feat the same way UNIQ does.

Step 5.2') Input Branches Form Tree? Check to see if the input branches form a tree. Let $A := [A_T | A_L]$ be the reduced node-incidence matrix of the controlled-source graph where A_T contains the columns describing the input port branches and A_L contains the columns describing the output port branches. The input port branches form a tree if and only if $\det A_T \neq 0$. The determinant is calculated using the algorithm described in the previous section.

Step 5.3') Calculate Δ. Δ is defined in terms of the fundamental cutset matrix $Q := [1 | Q_L]$. Internally, UNIQF uses the reduced node-incidence matrix so we need to calculate Δ in terms of A_T and A_L.

Fact 3: Let $[A_T | A_L]$ be the reduced node-incidence matrix of a controlled-source graph, G_0 where A_T and A_L are $n \times n$ matrices and A_T is nonsingular. Then

$$\Delta = \det A_T \det (A_T + A_L C)$$

Proof: Using KCL it is easy to show that $Q_L = A_T^{-1} A_L$. Hence

$$\begin{aligned}
\Delta &= \det (I + C A_T^{-1} A_L) \\
&= \det (C A_T^{-1} (A_T C^{-1} + A_L)) \\
&= \det C \det A_T^{-1} \det (A_T C^{-1} + A_L) \\
&= \det A_T^{-1} \det (A_T C^{-1} + A_L) \det C \\
&= \det A_T \det (A_T + A_L C)
\end{aligned}$$

where we have used the fact that $\det A_T$ is ± 1. EOP.

We already have $\det A_T$ from step 5.2') so all that remains is to find $\det (A_T + A_L C)$. The determinant is calculated recursively using minor expansion. The algorithm is similar to, but slightly more complicated than, the algorithm for calculating the determinant of a reduced node-incidence matrix. The

complication arises since a column with more than one non-zero entry does not imply a zero determinant, but results in an expansion with more than one minor for the column. For speed and memory considerations, the matrix $A_T + A_L C$ is never explicitly stored in memory, its entries are calculated from A_T, A_L and C whenever needed.

Shortcuts: To eliminate unnecessary calculations, a few shortcuts are implemented in UNIQF.

By Corollary 2.1 of [2], if $\sum_{\mu} \alpha_{\mu max} \leq 1.0$, and the Interconnection Assumption holds, the circuit has a unique solution for all valid circuit parameters. UNIQF checks this inequality after the Interconnection Assumption is checked and whenever a controlled-source graph is generated. If the inequality holds, UNIQF does not go deeper in the recursion. If the $\alpha_{\mu max}$'s are small, as in the case of transistor reverse current gains, this shortcut is very effective.

Like UNIQ, UNIQF checks the connectedness of the circuit at the beginning of analysis and after each resistor is opened and each controlled source zeroed. If, at any stage in the recursion, the circuit becomes unconnected, UNIQF stops further analysis on this branch of the recursion tree.

By Remark 2 of Theorem 2 in [2], UNIQF need only perform the determinant test on those graphs which have neither a self-loop nor a bridge. Since checking these conditions is usually much faster than a determinant test, UNIQF checks these conditions before every determinant test.

6. Additional Details

UNIQ and UNIQF were developed on a CompuPro 8086-based microcomputer running the PC-PRO implementation of the PC-DOS operating system from Computer House using the Computer Innovations C86 C compiler. Source code for the input parsing routines was generated on UNIX using the *yacc* and *lex* utilities. Both programs run on UNIX and on any PC-DOS or MS-DOS based machine (including the IBM PC, XT and AT).

The C programming language was chosen for three main reasons: C is portable, C allows recursion and C has flexible dynamic memory allocation facilities.

User's guides for UNIQ and UNIQF are in Appendices 4 and 5, respectively. For information on how to obtain copies of the two programs (both executable files and C source files) please contact the authors.

Appendix 1: Statement of Theorem 8

Let B be the fundamental loop matrix of a graph G. G exhibits *complementary tree structure* iff B has the form $[B_T \mid 1]$ where B_T is square and nonsingular. If $\det B_T > 0$ (resp. $\det B_T < 0$), then the graph is said to have positive (resp. negative) complementary tree structure. A graph consisting of a single node is defined to have positive complementary tree structure.

Let N be a circuit containing any combination of independent voltage and current sources, strictly increasing (and onto) two-terminal resistors and linear controlled sources (of all four types). Let G be the digraph associated with N. (Note that the direction of the branch associated with the output port of a CCCS or VCCS is defined *opposite* to the direction in the controlled-source symbol.)

A *controlled-source graph* G_o is obtained from G by the following three operations:

(a) zero each independent source

(b) either open- or short-circuit each resistor

(c') zero some (possibly none) of the controlled sources.

Furthermore, a controlled-source graph must satisfy the following constraint (Interconnection Assumption). If there are any loops (resp. cutsets) in N composed exclusively of independent voltage (resp. current) sources, output ports of CCVSs and VCVSs (resp. CCCSs and VCCSs) and input ports of CCVSs and CCCSs (resp. VCVSs and VCCSs), then G_0 must contain at least one branch from each of these loops (resp. cutsets).

Theorem 8 states that, for all independent source values, for all strictly increasing (and onto) two-terminal resistor characteristics and for all *positive* controlled-source coefficients, N possesses a unique solution if and only if there exists at least one controlled-source graph with *positive* complementary tree structure or at least one controlled-source graph with *negative* complementary tree structure, but not both.

When calculating complementary tree structure of a controlled-source graph G_0, we require the branches corresponding to the input ports to form one tree and the branches corresponding to the output ports the other. Hence, the following branch numbering scheme must be used. If G_0 contains k controlled sources, then number the input branch of the ith source as branch i, the output branch of the ith source as branch $k + i$.

Appendix 2: Statement of Theorem 2

We state Theorem 2 for the case of CCCSs. The dual statement holds for VCVSs. Theorem 2 does not apply to VCCSs or CCVSs.

Let N be a circuit containing any combination of independent voltage and current sources, strictly increasing (and onto) two-terminal resistors and k linear CCCSs whose controlling coefficients α_μ satisfy $0 < \alpha_\mu < \alpha_{\mu\max}$ for $\mu = 1, \ldots, k$. Let G be the graph associated with N. (Note that the direction of the branch associated with the output port of a CCCS is defined opposite to the direction in the controlled-source symbol.)

Define a controlled-source graph G_0 as in Appendix 1, but without the constraint imposed by the Interconnection Assumption.

Theorem 2 states that, for all independent source values, for all strictly increasing (and onto) two-terminal resistor characteristics and for all controlling coefficients α_μ satisfying $0 < \alpha_\mu < \alpha_{\mu\max}$ for $\mu = 1, \ldots, k$. N possesses a unique solution if and only if

1) N contains no loop consisting of independent voltage sources and input ports of CCCSs.

2) N contains no cutset consisting of independent current sources and output ports of CCCSs.

3') We cannot obtain a connected controlled-source graph G_0 such that $\Delta < 0$ where $\Delta := \det(I + CQ_L)$ where $C := \mathrm{diag}(\alpha_{\mu_1\max}, \ldots, \alpha_{\mu_k\max})$ and Q_L is the main part of the fundamental cutset matrix $Q := [1 | Q_L]$.

Appendix 3: Algorithms for Verifying the Interconnection Assumption

The Loop Restriction

The loop restriction imposed by the Interconnection Assumption may be reduced to the following problem. Given a graph G with node-incidence matrix A and a subset of branches $B := \{b_1, \ldots, b_k\}$ of G, find all of the loops of G composed solely of branches from B. Here is pseudo-code for a routine *find_loops*(A, B) which does just this.

```
begin find_loops(A, B)
    tag all branches not in B
    initialize loop list L to NULL
    for each branch b_i in B
        tag branch b_i
        untag all the nodes
        let n_1 be one of the nodes adjacent to b_i
        let n_2 be the other
        tag nodes n_1 and n_2
        find all allowed paths from n_1 to n_2
        append branch b_i to every such path
        add this list of loops to the loop list L
    return the loop list L
end find_loops()
```

This routine is fairly simple. To keep track of which branches are under consideration, *find_loops()* uses tags. If a branch is tagged, it cannot be used to construct a loop. After tagging the branches not in B, *find_loops()* processes the branches in B one at a time. For each branch b_i, all paths are found which connect the two nodes adjacent to b_i and which contain only untagged branches and pass through untagged nodes.

There are two important points here. First, a loop cannot pass through the same node twice so nodes are tagged as well as branches. Second, once branch b_i is tagged, it is never untagged. In the *ith* iteration, *find_loops()* finds all loops containing branch b_i with remaining branches in $\{b_{i+1}, \ldots, b_k\}$. Hence, the same loop is never found twice.

A recursive function *node_path*(n_1, n_2, A) is used to find the allowed paths between two nodes. Here n_1 and n_2 are the two nodes and A is the node-incidence matrix.

```
begin node_path(n1, n2, A)
    if n1 equals n2
        return empty list
    initialize path list P to NULL
    for each untagged branch bi
        if bi is incident to n2 and other node of bi, n3, is untagged
            tag branch bi
            tag node n3
            path = node_path(n1, n3, A)
            if path not NULL
                append bi to each entry in path
                add entries in path to the path list P
            untag branch bi
            untag node n3
    return the path list P
end node_path()
```

node_path() first checks that n_1 and n_2 are different. If not, an empty path list is returned. For each untagged branch incident to n_2 and some untagged node n_3, *node_path*() tags that branch, tags node n_3 and calls itself to find all paths between nodes n_1 and n_3. If paths are found, the path list P is updated.

Note that P is initially set to NULL. NULL is different from an empty list in that NULL indicates no paths were found while an empty list implies a path was found but it contains no branches (that is n_1 equals n_2).

The Cutset Restriction

The cutset restriction imposed by the Interconnection Assumption may be reduced to the following problem. Given a graph G with node-incidence matrix A and a subset of branches $B := \{b_1, ..., b_k\}$ of G, find all of the cutsets of G composed solely of branches from B. Here is pseudo-code for a routine *find_cutsets(A, B)* which does just this.

```
begin find_cutsets(A, B)
    initialize cutset list C to NULL
    short all branches in A which are not in B
    let no_comp be the number of components in A
    for each combination of branches C' in A
        copy A to A'
        open branches in A' which are in C'
        if the number of components in A' equals no_comp + 1 and
          no subset of C' is in the cutset list C
            add C' to the cutset list C
    return the cutset list C
end find_cutsets()
```

To eliminate unwanted branches, all branches not in the target set B are shorted. Remember that a cutset is a set of branches C such that 1) removal of the branches in C increases the number of components of the graph by one and 2) no subset of C possesses property 1). For each combination C' of branches of B, the branches specified by cutset candidate C' are opened and the number of components of the resulting graph is checked. If there is one more component than in the original graph and C' is not a superset of any cutset already in the list, C' is a cutset and is added to the list.

The cutset candidates C' are generated recursively with a function similar to *open_short()*. Instead of the two operations *open* and *short* applied to each resistor in the circuit, the operations *leave out of* C' and *include in* C' are applied to each branch in B. The order in which these operations are applied is very important. Let C_j' be the *jth* cutset candidate generated by the recursion. Then C_j' must not be a subset of C_i' for all $i < j$. This ordering can be achieved in the recursion by first *leaving out* a branch and then *including* the branch. If a different ordering is used, then *find_cutsets*() would not only need to check that no subset of C' is in the cutset list C, but also delete from C any supersets of C'.

Appendix 4: User's Guide for UNIQ

INSITE, NONLINEAR SYSTEMS GROUP, U.C. BERKELEY

<< UNIQ >> (1.1) - USER GUIDE

Michael Peter Kennedy
September, 1985

COMPUTER IMPLEMENTATION OF "THEOREM 8" (T. Nishi & L.O.Chua "Topological Criteria for Nonlinear Resistive Circuits Containing Controlled Sources to have a Unique Solution" IEEE Trans. Ccts. & Syst. CAS-31, No.8, Aug.1984).

STATEMENT OF THE THEOREM, WITH ASSUMPTIONS AND RESTRICTION

Let N be a general circuit, containing resistors (positive linear two-terminal resistors and/or two-terminal nonlinear resistors with monotone-increasing onto v-i characteristics), independent dc voltage and current sources, and linear controlled sources (with real positive controlling coefficients).

INTERCONNECTION ASSUMPTIONS

1. There is no LOOP in N composed exclusively of the following:
   a. DC voltage source(s).
   b. Output (controlled) edge(s) of CCVS or VCVS.
   c. Input (controlling) edge(s) of CCVS or CCCS.

2. There is no CUTSET in N composed exclusively of the following:
   a. DC current source(s).
   b. Output (controlled) edge(s) of CCCS or VCCS.
   c. Input (controlling) edge(s) of VCVS or VCCS.

For such a circuit N, there exists a unique solution for all circuit parameters if and only if by applying operations (a) (short-circuiting each independent voltage source and open-circuiting each independent current source), (b) (open- or short-circuiting each resistor (all permutations must be tried)), and (c') (zero some (possibly none) controlled sources) to the associated graph G, it is possible to obtain at least one graph with positive- or one with negative-complementary tree structure, but not both. (A one-node no-branch graph is deemed to have positive complementary tree structure).

In applying operation (c'), the following restriction must hold :

RESTRICTION

Suppose that in G there exist some loops and/or cutsets which violate the Interconnection Assumptions, then in applying operation (c'), we must ensure that the resulting controlled source graph contains at least one branch per violating loop or cutset.

INPUT

SPICE-like input format - no special file terminator is required.
Each circuit element is completely described by a single-line entry, consisting of three fields, as follow:

{NAME} {TOPOLOGY} {VALUE/RELATION}

1. NAME

A name field consists of a string of alphanumeric characters.
The first character of the name, which identifies the element type, must be a capital letter. This theorem deals with just seven different types of circuit elements, indicated thus :

E : voltage-controlled voltage source (VCVS)

F : current-controlled current source (CCCS)

G : voltage-controlled current source (VCCS)

H : current-controlled voltage source (CCVS)

I : independent current source

R : resistor (linear or nonlinear)

V : independent voltage source

If, while reading the input file, the program encounters an illegal element name (one not beginning with one of the seven valid characters listed above), then an error condition arises. The program indicates that an illegal element has been encountered, and will not proceed to analyse the circuit until the error in the input file has been corrected by the user.

2. TOPOLOGY

The topological field consists of either one (resistor or independent source) or two (controlled source) pairs of integers, denoting the terminal node numbers of the corresponding one- or two-port. Each port node pair is chosen such that the reference current flows IN to the port through the FIRST node and OUT by the SECOND node, with the FIRST node at a HIGHER reference potential than the second. In the case of a controlled source, the FIRST node pair describes the CONTROLLED PORT, while the SECOND refers to the CONTROLLING PORT. Thus, the topological fields for a resistor,

independent source and a controlled source are as follow:

RESISTOR :

```
{   (    in-current node    )   (    out-current node    )   }
    (  high-potential node  )   (   low-potential node   )
```

INDEPENDENT SOURCE :

```
{   (    in-current node    )   (    out-current node    )   }
    (  high-potential node  )   (   low-potential node   )
```

CONTROLLED SOURCE :

```
(          controlled port          ) (          controlling port          )
{( in-curr. node  ) ( out-curr. node ) ( in-curr. node  ) ( out-curr. node )}
 ( high-pot. node ) (  low-pot. node ) ( high-pot. node ) (  low-pot. node )
```

The reference direction for the controlled source graph edge associated with a port is from in- to out-current (high- to low-potential) nodes.

3. VALUE/RELATION

A value/relation field is used to characterise the terminal voltage-current relationship for a circuit element.

It is assumed that ALL RESISTORS (linear and nonlinear) are two-terminal elements characterized by STRICTLY MONOTONE INCREASING v-i curves. Both the value and relation are optional. If a relation (element charactistic, surrounded by brackets {} - we allow this to provide compatibility with other software in the INSITE suite)is found in this field, it is assumed to describe a nonlinear resistor of the assumed type; if a negative resistance value is found, a warning is printed, and the program stops once the entire input file has been scanned.

In the process of implementing the theorem, all independent sources are first zeroed; hence their value fields are of no significance, and so are ignored. In the case of a controlled source, it is assumed, without loss of generality, that EACH CONTROLLED SOURCE IS LINEAR WITH A POSITIVE REAL CONTROLLING COEFFICIENT (alpha). Therefore, the program looks in this field for a positive floating point number. If a non-positive number is found, a warning is printed, and program execution ceases once the entire input file has been scanned.

EXAMPLE

```
This is the title                    << FIRST LINE OF INPUT IS THE TITLE >>
                                           << BLANK LINE - IGNORED >>
* resistors                                  << COMMENT - IGNORED >>
R1    1   2     1.6                                 OK
R2    2   4     1k                                  OK
R3    3   1     2500                                OK
R4    5   2    -659.65                           ***** ERROR *****
RNL   1   4     { ... }                           << ASSUMED OK >>
R1g   7   2                                         OK
                                           << BLANK LINE - IGNORED >>
V4    2   5    -4.76                               OK
V56   2   7                                         OK
V5    1   4     5k87                              OK
V8    3   5     { ... }                           OK
Vg    3   6     23                                OK
                                           << BLANK LINE - IGNORED >>
I8    1   2     2                                 OK
Iop   3   6    -8.976                             OK
Ipr   5   7     { ... }                           OK
Ifh   6   4                                       OK
I     2   5     5u3                               OK
* another comment                            << COMMENT - IGNORED >>
F1    4   1     3   4     1.5                     OK
FR    4   2     4   1     3                       OK
HE    3   5     2   1    -5.671                 ***** ERROR *****
Egh   4   3     1   5     { ... }                ***** ERROR *****
G     4   5     5   2     6k2                     OK
Fz    3   5     3   2                          ***** ERROR *****
```

GETTING STARTED WITH << UNIQ >>

1. Create a SPICE-compatible input file, called "in.dat" for example, describing the circuit to be analysed.
Remember :
    (a) only MONOTONE-INCREASING RESISTORS are allowed. A negative linear resistor should be replaced by a controlled source.
    (b) controlled sources must be governed by a LINEAR relation with real positive controlling coefficient alpha. If alpha is negative, a warning to this effect will be printed and the circuit will not be analysed until the error has been corrected.
    (c) the topological field for controlled sources should be given in STANDARD SPICE-compatible form.

2. Run "UNIQ" - usage : UNIQ [-v] [filename.extension]
Input can be from a file selected by the user by including the filename (with extension) in the command line argument list.
If a filename is not specified, input is read from stdin.
If keyboard entry is desired, no filename should be given, and when the program title appears on the screen, enter the circuit data exactly as one would if writing to a data file, one element per line, each line terminated by a "carriage return". Type "<control> Z" (^Z) to denote "end of file".

```
Type :          UNIQ  in.dat          << COMMAND LINE FORM >>
or :            UNIQ                    << KEYBOARD INPUT >>
```

The results will be directed to "standard output", typically the monitor screen. Alternatively, to generate a "hard-copy" of the analysis, the output can be redirected to the printer as follows.

```
Type :          UNIQ  in.dat > prn << COMMAND LINE IN, HARD-COPY OUT >>
or :            UNIQ         > prn   << KEYBOARD IN, HARD-COPY OUT >>
```

The output may also be directed to an output file, say "out.dat".

```
Type :          UNIQ  in.dat > out.dat  << COMMAND LINE IN, REDIRECT OUT >>
or :            UNIQ         > out.dat    << KEYBOARD IN, REDIRECT OUT >>
```

3. VERBOSE MODE
UNIQ provides two output modes, normal (default) and verbose (selected by including the optional -v mode selector on the command line argument list). In verbose mode, for EVERY reduced controlled source graph tested, the program gives information about the state of each resistor (open- or short-circuited) and controlled source (intact or zeroed), as well as the edge directions and terminal connections for each INTACT controlled source, and indicates which type of complementary tree structure (positive or negative), if any, the graph possesses. (In normal (default) mode, this information is given only for each "new" complementary tree structure encountered.) Each time the interconnection restrictions are tested, lists of the potential loop- and cutset- restriction-violating controlled source edges are printed (verbose mode only).

OUTPUT

The output has been chosen to maximise the quantity of useful information provided to the user, while minimising the total output.

First, a title is printed, introducing the program :

UNIQ 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

If verbose mode has been selected, a message to this effect is printed to standard output (typically the terminal screen).

UNIQ then reads the specified input file, echoing first the title and then a copy of the names and terminal connections of each of the elements in the circuit under analysis, as they appear in the input file.

Next, the program checks for loops of just voltage sources and cutsets of just current sources. If one of either is found, a warning is printed, along with a list of the names of the elements in the violating set(s), before the program terminates, outputting its conclusion that the circuit does NOT have a unique solution for all parameters.

If such a pathological case does not exist, UNIQ checks for loops and cutsets which violate the Interconnection Assumptions. If any are found, a list of the named branches in each of the illegal sets is printed.

Next, independent sources are zeroed, (a message to this effect is printed to standard output if verbose mode has been selected) and operations (b) (open- or short-circuit each resistor) and (c') (leave intact or zero each controlled source) are carried out.

When complementary tree structure (positive or negative) is found, of a sign which has not already been encountered during program execution, the interconnection assumption restriction is checked, and if verbose mode has been selected, a copy of the names of the potential loop- or cutset-violating elements is printed.

If the restriction holds, information is given about the state of each resistor (whether OPEN- or SHORT-circuited) and controlled source (whether INTACT or ZEROED), and UNIQ indicates the type of complementary tree structure found. For each INTACT controlled source, the terminal connections and directions (as specified by Theorem 8) of the input and output edges are also given. With this information, one can reconstruct the reduced controlled source graphs and in doing so, see how one might, by reversing an edge, say, force a "non-unique" circuit to have a unique solution (see EXAMPLE below).

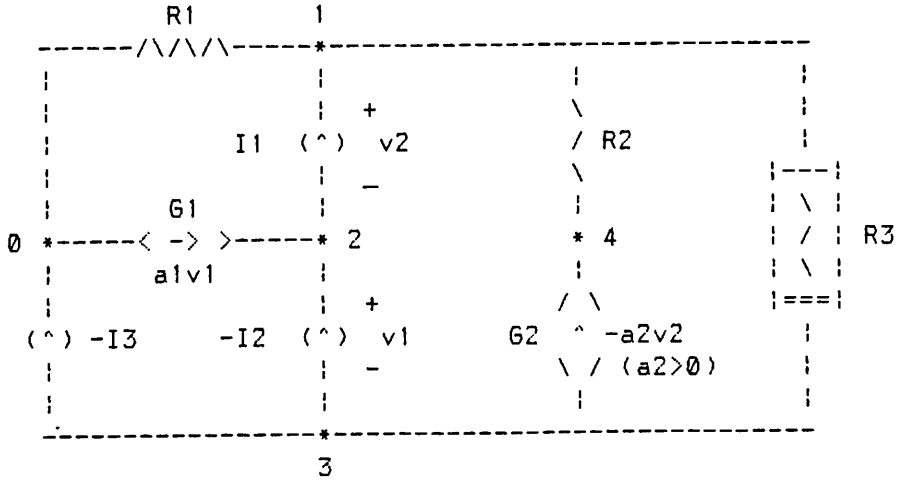
Testing continues until both negative and positive complementary tree structure are found, or all reduced controlled source graphs which have complementary tree structure have been found and checked.

Finally, the conclusion is reached as to whether or not the circuit has a unique solution for all parameters, and the result printed to stdout.

EXAMPLE (fig6(d), Nishi & Chua)

The SPICE-like description of the circuit of figure 1 is shown below (<< EXAMPLE - INPUT FILE >>). With this input, UNIQ finds the illegal cutset composed of the input and output branches of G1, the input edge of G2, and the independent current sources I1 and I2. Running the program in verbose mode, we find that we can obtain only the complementary tree structure graphs of figs. 2(a) and (b) (reconstructed from data in the output file (<< EXAMPLE - OUTPUT FILE >>), both of which have negative complementary structure. Hence, UNIQ concludes that this circuit has a unique solution.
However, if the direction of v2 is reversed, then both positive (fig.3a) and negative (fig.3b) complementary tree structure graphs can be found (reconstructed from output file (<< EXAMPLE (v2 REVERSED) - OUTPUT FILE), and UNIQ concludes that the solution is NOT unique.

Figure 1



<< EXAMPLE - INPUT FILE >>

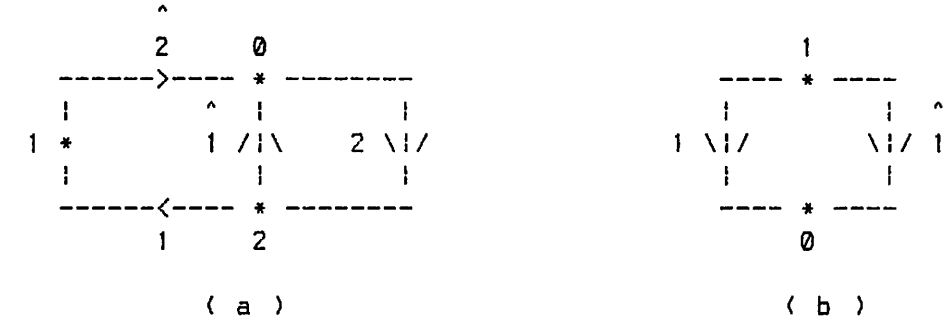
```
fig 6(d)

R1   0 1
R2   1 4
R3   3 1

I1   2 1
I2   2 3
I3   0 3

G1   0 2   2 3   1
G2   4 3   1 2   1
```

Figure 2



<< EXAMPLE - OUTPUT FILE >>

UNIQ 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: fig 6(d)

```
        R1    0   1
        R2    1   4
        R3    3   1
        I1    2   1
        I2    2   3
        I3    0   3
        G1    0   2    2   3
        G2    4   3    1   2
```

Cutsets which violate the interconnection assumption:
1) G1 in G1 out G2 in I1 I2

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	short
R2	short
R3	open

Controlled Sources	State	Input Branch	Output Branch
G1	intact	2 -->-- 1	2 -->-- 0
G2	intact	0 -->-- 2	1 -->-- 0

Graph has NEGATIVE complementary tree structure

****

only negative complementary
tree structure exists

UNIQUE SOLUTION EXISTS

****

<< EXAMPLE (v2 REVERSED) - INPUT FILE >>

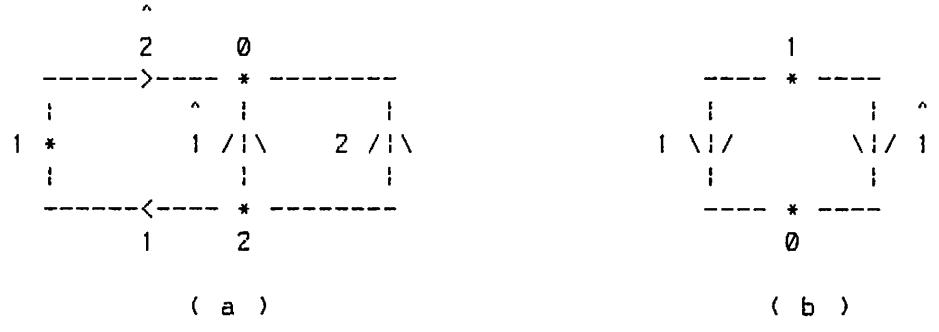
```
fig 6(d)

R1  0 1
R2  1 4
R3  3 1

I1  2 1
I2  2 3
I3  0 3

G1  0 2  2 3  1
G2  4 3  2 1  1
```

Figure 3



<< EXAMPLE (v2 REVERSED) - OUTPUT FILE >>

UNIQ 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: fig 6(d)

R1	0	1		
R2	1	4		
R3	3	1		
I1	2	1		
I2	2	3		
I3	0	3		
G1	0	2	2	3
G2	4	3	2	1

Cutsets which violate the interconnection assumption:
1) G1 in G1 out G2 in I1 I2

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	short
R2	short
R3	open

Controlled Sources	State	Input Branch	Output Branch
G1	intact	2 -->-- 1	2 -->-- 0
G2	intact	2 -->-- 0	1 -->-- 0

Graph has POSITIVE complementary tree structure

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	short
R2	short
R3	short

Controlled Sources	State	Input Branch	Output Branch
G1	intact	1 -->-- 0	1 -->-- 0
G2	zeroed		

Graph has NEGATIVE complementary tree structure

****

both positive and negative
complementary tree structures exist

SOLUTION IS NOT UNIQUE

****

ERROR CONDITIONS - WHEN THEY OCCUR AND HOW TO CURE THEM

1. Cannot open input file

This error occurs if the input file specified on the command line argument list cannot be opened (because the name has been incorrectly spelled, or the file does not exist). Check that the file exists in the current directory (or that the pathname has been correctly specified if the selected input file resides in another directory), and that the file name and extension have been correctly specified.

2a. negative resistance not allowed
 b. negative coefficient not allowed
 c. nonlinear controlled source not allowed

These occur when a negative resistor, linear controlled source with negative controlling coefficient, or nonlinear controlled source (respectively) is encountered in the input file. Being an error condition, this will cause the program to abort, but only after the entire input file has been read (to allow for the detection of multiple errors).

3. Too many nodes (N maximum)

If the input circuit contains more than N nodes, it cannot be analysed. In this case, the program aborts.

4. Too many branches (M maximum)

If the input circuit contains more than M branches, it cannot be analysed. In this case, the program aborts.

5. Error in input data

This occurs when a SPICE element record is incomplete or contains wrong variable types (i.e. those whose names begin with other than E,F,G,H, I,R, or V). Once again, this is an error condition, causing the program to abort after the entire input file has been read.

6. No elements in input file

If, after reading the specified input data file, no valid circuit elements have been found, the program aborts, since it has no data to process. Check that the correct input file has been specified.

7. Program fault

If the computer hardware and << UNIQ >> program are functioning correctly, this message can never be generated. It is included as the default case on all internal "switch( )" statements.

8. Duplicate element name

This occurs when two or more SPICE-compatible element records have the same name(s). This is an error condition, causing the program to abort only after the entire input file has been read, once again allowing for correction of multiple errors after just one pass of the input file.

9. Out of memory

This error indicates that an attempt to allocate some storage space has been unsuccessful. This is an ABORT condition.

10. Usage: uniq [-v] [file]

This message is printed when too many arguments are given on the command line.

FOR THE SERIOUS USER, A LITTLE MORE DETAIL

THE ALGORITHM ...

Create structures p (of type RNI), c (of type CUT_LIST), and l (of type LOOP_LIST) which will contain the node incidence matrix of graph G (and associated book-keeping overhead), linked list of cutsets violating the interconnection assumptions (if any), and linked list of illegal loops (if any), respectively.

Print title.

Reset flags - all flags are initially reset to FALSE. These are used to indicate the types of tree structure found to date, whether or not there exist any loops or cutsets in the original graph G which violate the interconnection assumptions, whether or not "verbose" mode has been selected by the user, and whether or not there exist in G loops of independent voltage sources only and/or cutsets of independent current sources only. Flag[0] (PCTS) is set TRUE if and when "valid" (i.e. Interconnection restriction satisfied) positive complementary tree structure is found; flag[1] (NCTS) is similarly set TRUE for negative complementary tree structure. Flag[2] (BAD_CUTS) is TRUE if interconnection-assumption-violating cutsets exist; flag[3] (BAD_LOOPS) similarly indicates illegal loops. Flag[4] (VERBOSE) is set TRUE if the user selects "verbose" mode by typing -v on the command line. Flag[5] (IVS_LOOPS) is set TRUE if G contains one or more loops of just independent voltage sources; flag[6] (ICS_CUTS) is set TRUE if G contains one or more cutsets of just independent current sources.

Select mode - by scanning the command line argument list, the program determines whence comes its input (setting file pointer fp appropriately (default : stdin)), and whether or not "verbose" mode is desired. If verbose mode is selected, flag[4] (VERBOSE) is set, and a message to this effect printed to stdout.

Read SPICE-compatible input file indicated by fp (a data file (with extension) specified in the command line argument list, or redirected to standard input using the "<" input specifier, or data read directly from the keyboard), renumbering the nodes from zero (the first node encountered at the input becomes "node zero"; the next one is "node one", and so on), and reordering branches in the RNI_MATRIX, using the template

(VCVS,CCCS,VCCS,CCVS ¦ VCVS,CCCS,VCCS,CCVS ¦¦ R ¦¦ IVS ¦ ICS),

controlled source controlling edges, controlled edges, resistors, independent voltage sources, and independent current sources. If a disallowed element (including negative linear resistors) is found, or the controlling coefficient (alpha) of a controlled source is negative, a warning is printed to stdout, and the routine aborts after reading

through to the end of the input file ; this allows for correction of multiple errors following just one pass of the input data file.

[Note that Theorem 8 requires that the output edge associated with a CONTROLLED CURRENT SOURCE be directed OPPOSITE to the arrowhead inside the diamond-shaped symbol. Therefore, when UNIQ forms the RNI_MATRIX, it takes standard SPICE-like input and converts it to the non-standard form required here. Thus, the first node of the SPICE input for a controlled current source is used by the program as the "negative" terminal and the second as the "positive" terminal of the output edge.]

At this point, three string arrays, r_name, cs_name and is_name, are created. These contain the element names of each resistor, controlled source, and independent source, and are used in referring to the elements, e.g. when the program finds a complementary tree structure, to indicate BY NAME which resistors have been open-circuited, and which have been short-circuited, as well as which controlled sources are still present in the corresponding reduced controlled source graph.

Next, two structures R_STATE (resistor state) and CS_STATE (controlled source stste) are generated, and tagged on to RNI structure p. R_STATE contains a character array "state" of dimension equal to the number of resistors in the master copy of the circuit under analysis. Each element in the array contains either an OPEN ('o') or a SHORT ('s') character, to denote an open- or short-circuit condition respectively. Similarly, CS_STATE contains a character array "state" of dimension equal to the number of controlled sources in the master copy of the circuit under analysis. Each element in this array contains either an INTACT ('i') or ZEROED ('z') character, to indicate whether or not the associated source is still present in the reduced controlled source graph. This information is used when checking, at the core of the program, whether or not sufficient loop and cutset edges have been retained to satisfy the interconnection assumption restrictions.

Check interconnection Assumptions, i.e. that there exist no loops made exclusively of dc voltage sources, CCVS or VCVS output edges, and CCVS or CCCS input edges, and no cutsets made exclusively of dc current sources, CCVS or VCCS output edges, and VCVS or VCCS input edges.
First, the program checks for "voltage source only loops" and "current source only cutsets". If one of either can be found, the circuit does not have a unique solution, so the appropriate flags (IVS_LOOPS and/or ICS_CUTS) are set, the result printed, and execution ceases.
Next, the program finds all loops and/or cutsets which violate the interconnection assumptions, storing them in two linked lists l and c. The illegal sets (if any) are printed to stdout. This is the "user's" list of illegal sets. The program itself acts upon a subset from which the independent sources have been removed.

Perform operation (a) - Zero all independent sources, i.e. open-circuit all independent current sources and short-circuit all independent voltage sources. This function is performed by subroutine zero_is( ).

Finally, the program generates two lists of illegal loops and cutsets respectively containing only controlled source edges. If at least one illegal cutset is found, flag[2] (BAD_CUTS) is set TRUE and the cutsets stored in linked list c. If at least one illegal loop is found, flag[3] (BAD_LOOPS) is set TRUE and the violating loops stored in linked list l.

If there were independent sources in the circuit under analysis, memory would have been allocated for the array is_name; this memory is now freed.

Check whether or not the circuit contains controlled sources. If it does not, then the network is purely resistive and may be reduced to a one-node positive complementary tree structure graph by shorting all resistors, if and only if the graph is made of just one piece; if the graph is unconnected, a one-node graph cannot be found, so the solution is not unique. If the network is found to be purely resistive at this stage, the conclusion on uniqueness of solution is printed and program execution ceases.

Perform operation (b) - Recursively open and short each resistor in turn to generate all possible permutations of "controlled source only" graphs. This operation is carried out by open_short( ). If, by opening any resistor, the graph becomes disconnected, then a controlled source graph with complementary tree structure will not be found, so the program discontinues its search along the current path, and returns up the recursive tree.

For each "controlled source only" graph thus generated, perform operation (c'), i.e. recursively zero or not each controlled source in turn, producing all possible permutations of reduced controlled source graphs. This is performed by zero_s( ). If, at any stage, there are fewer branches than nodes, further zeroing of sources can yield at best positive complementary tree structure. If PCTS is TRUE, indicating that positive structure has already been found, then further zeroing will provide no more information. Also, if the graph becomes unconnected, then a complementary tree structure will not be found by further zeroing, so recursion along the present path ceases (these are just computation reduction tricks). If operation (c') has been performed in its entirety (each controlled source in turn has been zeroed or not), or if a one-node graph now exists, then test for complementary tree structure.

For each resulting reduced controlled source graph, test for complementary tree structure, positive or negative. If either type is found, which has not already been encountered, check that the

interconnection assumption restrictions are satisfied (this is another computation reduction trick - we must check for complementary tree structure for each reduced controlled source graph, but need only test the interconnection assumption restriction if a "new" type of structure has been found). If so, the appropriate complementary tree structure flag is set TRUE (flag[0] (PCTS) for positive, flag[1] (NCTS) for negative). If the interconnection restrictions are not satisfied, the flags remain unchanged. If, at this point, both flag[0] (PCTS) and flag[1] (NCTS) are TRUE, indicating that valid positive and negative complementary tree structures have been found, the program outputs its verdict that the solution is not unique, and terminates. Otherwise, it continues to test "all", within the constraints already outlined, remaining possible reduced controlled source graphs for complementary tree structure, eventually returning to the main program with no more than one of flags PCTS and NCTS set TRUE. If neither flag is set, the solution is not unique; if either flag[0] (PCTS) or flag[1] (NCTS) is TRUE, the solution is deemed unique. If "verbose" mode has been selected, then each time a graph is tested for complementary tree structure, the states of all resistors (OPEN- or SHORT-circuited) and controlled sources (INTACT or ZEROED) are printed to standard output.

The interconnection assumption condition is tested as follows. For the reduced controlled source graph, it is checked that at least one branch has been retained per illegal cutset/loop. If neither illegal loops nor cutsets exist, ica_ok( ) returns TRUE. If illegal loops or cutsets exist and no controlled source edges remain in the graph, then ica_ok( ) returns FALSE (interconnection restriction NOT satisfied). Since the illegal cutset and loop elements are labelled according to original branch number, we must first form a list of the original branch numbers of the edges remaining in the controlled source graph under test. This is done by extracting the INTACT branches from the CS_STATE structure, and forming a CS_ID (controlled source identifier) structure cs, containing a list of the original branch numbers of the edges in the reduced controlled source graph under test. If at least one illegal cutset { loop } exists (flag[2] (BAD_LOOPS) TRUE) { (flag[3] (BAD_CUTS) TRUE) }, then extract from CS_ID structure cs those remaining controlled source branches (if any) which could possibly be in violating cutsets { loops }, (these are written to other CS_ID structures csc { csl }) and check that at least one such branch has been retained per illegal cutset and loop. If either of these conditions is not satisfied, ica_ok( ) returns FALSE.

Appendix 5: User's Guide for UNIQF

UNIQF

USER GUIDE

INSITE - NONLINEAR SYSTEMS GROUP, U.C. BERKELEY

COMPUTER IMPLEMENTATION OF "THEOREM 2" (T. Nishi & L. O. Chua "Uniqueness of Solution for Nonlinear Resistive Circuit Containing CCCS's or VCVS's whose Controlling Coefficients are Finite", ERL Memorandum No. UCB/ERL M84/88, 23 October, 1984).

STATEMENT OF THE THEOREM

Let N be a circuit containing any combination of independent voltage and current sources, strictly increasing (and onto) two-terminal resistors and k linear CCCSs or VCVSs (but NOT both) with finite positive controlling coefficients.

INTERCONNECTION ASSUMPTIONS

1. There is no LOOP in N composed exclusively of the following:

a. independent voltage source(s).

b. Output (controlled) branch(es) of VCVS or input (controlling) branch(es) of CCCS.

2. There is no CUTSET in N composed exclusively of the following:

a. independent current source(s).

b. Output (controlled) branch(es) of CCCS or input (controlling) branch(es) of VCVS.

Define the following operations on the circuit:

a. short-circuit all independent voltage sources and open-circuit all independent current sources.

b. open- or short-circuit all resistors, all permutations must be tried.

c. zero some (possibly none) controlled sources.

Then, define for the controlled source graphs, the determinant D

$$D = \det (I + C\ Q0),$$

where I is identity matrix, C is a diagonal matrix containing the controlling coefficients of the controlled sources associated with the controlled source graph which are resulted from performing operations a., b. and c. on the circuit N, and Q0 is the main part of the fundamental cutset matrix (resp. fundamental loop matrix) $Q = (I \mid Q0)$ of the reduced controlled source graph for CCCS (resp. VCVS) circuit.

The circuit has a unique solution for all valid circuit parameters if and only if by applying operations (a), (b) and (c) defined above, we can not obtain a connected controlled source graph such that

$$D = \det (I + C\ Q0) < 0.$$

INPUT

SPICE-like input format - the first line of the input file is the title and the last line is ".END" to indicate at the end of the file. Each circuit element is completely described by a single-line entry, consisting of three fields, as follows:

NAME TOPOLOGY VALUE/RELATION

1. NAME

A name field consists of a string of alphanumeric characters. The first character of the name, which identifies the element type, must be a capital letter. This theorem deals with just five different types of circuit elements, indicated thus :

I : independent current source

V : independent voltage source

R : resistor (linear or nonlinear)

E : voltage-controlled voltage source (VCVS)

F : current-controlled current source (CCCS)

If, while reading the input file, the program encounters an illegal element name (one not beginning with one of the five valid characters listed above or more than one type of controlled source is encountered), the program indicates that an illegal element has been encountered, and proceeds to check the rest of the input file for other errors (if any) and will not proceed to analyse the circuit until the errors in the input file have been corrected by the user.

2. TOPOLOGY

The topological field consists of either one (resistor or independent source) or two (controlled source) pairs of integers, denoting the terminal node numbers of the corresponding one- or two-port. Each port node pair is chosen such that the reference current flows into the port through the first node and out by the second node, with the first node at a higher potential than the second. In the case of a controlled source, the first node pair describes the controlled port, while the second refers to the controlling port. For each port, same rule applies as for two-terminal elements. Thus, the topological fields for a resistor, independent source and a controlled source are as follow:

RESISTOR :

(in-current node) (out-current node)

INDEPENDENT SOURCE :

(in-current node) (out-current node)
(high-potential node) (low-potential node)

CONTROLLED SOURCE :

(controlled port) (controlling port)

3. VALUE/RELATION

A value/relation field is used to characterize the constitutive relationship for a circuit element.

RESISTORS : It is assumed that all resistors are two-terminal elements characterized by passive strictly increasing v-i curves. Both the value and relation are optional. If a relation (element charactistic, surrounded by brackets {}) is found in this field, it is assumed to describe a nonlinear resistor of the assumed type; if a negative resistance value is found, a warning is printed, and the program stops once the entire input file has been scanned.

INDEPENDENT SOURCES : In the process of implementing the theorem, all independent sources are first zeroed; hence their value fields are of no significance and are ignored.

CONTROLLED SOURCES : it is assumed, without loss of generality, that each controlled source is linear and has been specified by a positive real controlling coefficient. Therefore, the program looks in this field for a positive floating-point number. If a non-positive number is found, a warning is printed, and program execution ceases once the entire input file has been scanned.

EXAMPLE :

```
        R1 0 3 1.000000
        R1 7 3 100.0
element R1 : duplicate element name
        F9 3 2 1 5 {}
element F9 : nonlinear controlled source not allowed
        F10 2 3 1 6 -0.990000
element F10 : negative coefficient not allowed
        H11 3 1 2 7 0.400000
element H11 : illegal element
        E12 2 1 3 8 0.400000
element E12 : only one type of controlled source is allowed
        V1 1 4 1.000000
        I0 1 5 {}
```

HOW TO RUN THE PROGRAM

1. Create a SPICE-compatible input file, called "in.dat" for example, describing the circuit to be analysed.

2. Run "UNIQF" - usage : UNIQF [-v] [-s] [filename]

a. Input

Input can be from a file selected by the user by including the filename. If a filename is not specified, input is read from stdin. If keyboard entry is desired, no filename should be given, and when the program title appears on the screen, enter the circuit data exactly as one would if writing to a data file, one element per line, each line terminated by a "carriage return". Type ^Z to denote "end of file".

b. Verbose Mode

In verbose mode, (selected by the user by a -v option in the command line) for every reduced controlled source graph tested, the program gives information about the state of each resistor (open- or short-circuited) and controlled source (intact or zeroed), as well as the terminal connections for each intact controlled source, and prints out the determinant value the graph possesses. In normal (default) mode, this information is given only when a negative determinant value,

which indicates the solution may not be unique, is found. The program exits with a negative message (indicating the solution is not unique) whenever it founds a case which gives a negative determinant or the program exits with a positive message (there exists a unique solution) after all the controlled source graphs have been checked and no negative determinant is found. In verbose mode, the program will go on and check all the determinants even when a negative determinant is found.

c. Save Mode

In save mode (selected by the user by -s option in the command line), when a negative determinant is found, the program asks for a file name to save the histroy of the operation, which specifies whether a resistor is opened or shorted and which controlled sources are zeroed (if any). The saved histroy information is useful to draw a controlled source graph on the color monitor. For more information, see documentation for program "csgraph" csgraph.doc.

d. Output

The results will be directed to "standard output", typically the monitor screen. Alternatively, to generate a "hard-copy" of the output, the output can be redirected to the printer by the following command line :

```
        UNIQF  in.dat > prn

or      UNIQF         > prn
```

The output may also be directed to an output file, say "out.dat".

```
        UNIQF  in.dat > out.dat

or      UNIQF         > out.dat
```

The output has been chosen to maximize the quantity of useful information provided to the user, while minimizing the total output.

UNIQF first reads the specified input file, echoing first the title and then a copy of element names and their terminal connections and the controlling coefficients of controlled sources as they appear in the input file.

Next, the program checks for loops and cutsets which violate the interconnection assumptions. If any are found, a list of the illegal sets is printed out by element names, and program exits after giving a negative answer.

Next, independent sources are zeroed, a message to this effect is printed out if verbose mode has been selected. and operations b. (open- or short-circuit each resistor) and c. (leave intact or zero

each controlled source) are carried out.

When a determinant (with negative value only for non-verbose mode) is calculated, information is given about the state of each resistor (whether opened or shorted) and controlled source (whether intact or zeroed, if intacted, the controlling coefficient is also given), and the determinant value is printed.

Testing continues until a conclusion is reached when a negative determinant is found or all combination of reduced controlled source has been checked, the program prints out the answer to stdout.

ERROR MESSAGES

We provide a list of error messages and interpret their meaning in this section. Hopefully, this will give the users a clue on how the errors happened and how to cure them.

1. Usage: uniqf [-sv] [file]

This message is printed when the command line entered does not match "uniqf [-sv] [file]" or "uniqf [file] [-sv]" where -sv could be replaced by either -vs, -v or -s.

2. Error opening file

This error occurs if the input file specified on command line cannot be opened. Check that if the file specified exists in current directory (or that the pathname has been correctly specified if the selected input file resides in another directory), and that the file name and extension have been correctly specified.

3. Error in input data

Error(s) found in input data meanning that either some syntax errors are found in input data or the circuit element does not satisfy the specification required by the Theorem.

4. Error closing file

This error occurs when finishing reading the input file or having saved the history file, the file is not successfully closed. Check if the disk is full when the latter happens.

5. VCCS is disallowed

Since UNIQF only deal with circuits contain CCCS or VCVS (not both), when an H element (VCCS) is encountered in the input file, this error

occurs.

6. CCVS is disallowed

Since UNIQF only deal with circuits contain CCCS or VCVS (not both), when a G element (CCVS) is encountered in the input file, this error occurs.

7. only one type of controlled source is allowed

This error occurs when more one type of controlled source is encountered in the input file.

8. nonlinear controlled source not allowed

This error occurs when controlled source is not specified by coefficient.

9. negative coefficient not allowed

This error occurs when controlled source is specified by a negative coefficient. Check if this is in error, if not, reverse either input or output port of the controlled source in question and change the sign of the coefficient.

10. duplicate element name

This occurs when two or more elements have the same names.

11. Out of memory

This error indicates that an attempt to allocate some storage space has been unsuccessful. Check if the size of circuit under test is too big.

12. No element in input file

If, after reading the specified input data file, no allowed circuit elements have been found, the program exits. Check if correct file name has been specified.

13. Too many nodes (64 maximum)

If the input circuit contains more than 64 nodes, it cannot be analysed due to the stack size of the computer. In this case, the program exits.

14. Too many branches (128 maximum)

If the input circuit contains more than 128 branches, it cannot be analysed due to the stack size of the computer. In this case, the program exits.

15. Wrong branch number

This will never happen under normal circumanstance --- We used this error code to debug UNIQF. Howerver, if it does happen, most likely it means the computer has run out of stack.

16. negative resistance not allowed

A resistor with negative resistance has been found in input file. Check if this is in error, if not, replace the resistor by a controlled source of approparite type.

17. Unknown error

Again, this should never happen under normal circumanstance --- We used this error code to debug UNIQF. Howerver, if it does happen, most likely it may suggest some thing wrong in the error-checking routines of UNIQF.

References

1. T. Nishi and L. O. Chua, "Topological Criteria for Nonlinear Resistive Circuits Containing Controlled Sources to have a Unique Solution," *IEEE Trans. Circuit Syst.*, vol. CAS-31, no. 8, pp. 722-741, August 1984.
2. T. Nishi and L. O. Chua, "Uniqueness of Solution for Nonlinear Resistive Circuits Containing CCCSs or VCVSs whose Controlling Coefficients are Finite," *IEEE Trans. Circuit Syst.*, to appear.
3. T. Nishi and L. O. Chua, "Non-Linear Op-Amp Circuits: Existence and Uniqueness of Solution by Inspection," *Int. J. Cir. Theor. Appl.*, vol. 12, pp. 145-173, 1984.
4. R. O. Nielsen and A. N. Willson, Jr., "A Fundamental Result Concerning the Topology of Transistor Circuits with Multiple Equilibria," *Proc. IEEE*, vol. 68, pp. 196-208, February 1980.

Figure Captions

Fig. 1. Circuit for Example 1.

Fig. 2. Listing of UNIQ output for Example 1.

Fig. 3. (a) is the circuit for Example 2; (b) and (c) are the two negative complementary tree structures which exist for this circuit.

Fig. 4. Listing of UNIQ output for (a) Example 2 (verbose mode) and (b) Example 3.

Fig. 5. Circuit for Example 4.

Fig. 6. Listing of UNIQF output for Example 4 with (a) $\alpha_1 + \alpha_2 = 1.001$ and (b) $\alpha_1 + \alpha_2 = 0.999$.

Fig. 7. Circuit for Example 5.

Fig. 8. Listing of UNIQF output for Example 5 with (a) $\alpha_1 = 1.001$ and (b) $\alpha_1 = 0.999$.

Fig. 9. Recursion tree showing the effect of *open_short*() on two resistors.

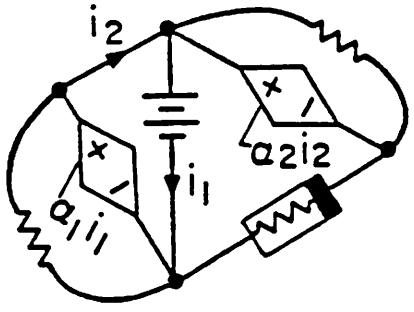
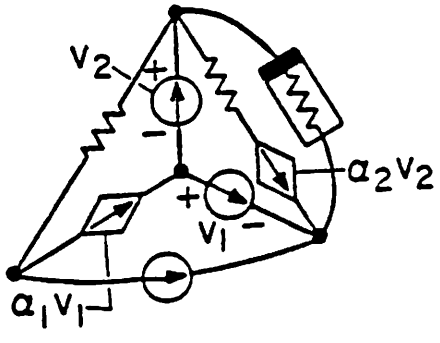
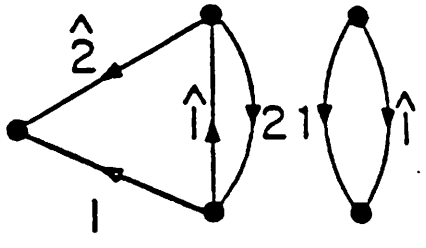


Fig 1



(a)



(b) (c)

Fig 3

UNIQ 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: Example 1

```
R1    1   2
R2    3   5
R3    1   5
V1    3   4
H1    2   1    4   1
H2    3   5    2   3
```

Loops which violate the interconnection assumption:
1) H1 in H1 out H2 in V1

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	open
R2	open
R3	open

Controlled Sources	State	Input Branch	Output Branch
H1	intact	1 -->-- 0	1 -->-- 0
H2	zeroed		

Graph has NEGATIVE complementary tree structure

****

only negative complementary
tree structure exists

UNIQUE SOLUTION EXISTS

****

Fig 2

UNIQ 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: Example 2

```
R1    0   1
R2    1   4
R3    3   1
I1    2   1
I2    2   3
I3    0   3
G1    0   2    2   3
G2    4   3    1   2
```

Loops of independent voltage sources:
none

Cutsets of independent current sources:
none

Loops which violate the interconnection assumption:
none

Cutsets which violate the interconnection assumption:
1) G1 in G1 out G2 in I1 I2

Zeroing Independent Sources

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	short
R2	short
R3	open

Controlled Sources	State	Input Branch	Output Branch
G1	intact	2 -->-- 1	2 -->-- 0
G2	intact	0 -->-- 2	1 -->-- 0

Graph has NEGATIVE complementary tree structure

potential illegal cutset branches still present:
G1 in . G1 out G2 in G2 out

Cutset restriction satisfied

Interconnection assumption satisfied

(output continued on next page)

Fig. 4(a)

(Example 2 continued)

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	short
R2	short
R3	short

Controlled Sources	State	Input Branch	Output Branch
G1	intact	1 -->-- 0	1 -->-- 0
G2	zeroed		

Graph has NEGATIVE complementary tree structure

potential illegal cutset branches still present:
G1 in G1 out

Cutset restriction satisfied

Interconnection assumption satisfied

****

only negative complementary
tree structure exists

UNIQUE SOLUTION EXISTS

****

Fig. 4(a) cont.

UNIQ 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: Example 3

```
R1    0   1
R2    1   4
R3    3   1
I1    2   1
I2    2   3
I3    0   3
G1    0   2    2   3
G2    4   3    2   1
```

Cutsets which violate the interconnection assumption:
1) G1 in G1 out G2 in I1 I2

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	short
R2	short
R3	open

Controlled Sources	State	Input Branch	Output Branch
G1	intact	2 -->-- 1	2 -->-- 0
G2	intact	2 -->-- 0	1 -->-- 0

Graph has POSITIVE complementary tree structure

STATES OF CIRCUIT ELEMENTS:

Resistors	State
R1	short
R2	short
R3	short

Controlled Sources	State	Input Branch	Output Branch
G1	intact	1 -->-- 0	1 -->-- 0
G2	zeroed		

Graph has NEGATIVE complementary tree structure

****

both positive and negative
complementary tree structures exist

UNIQUE SOLUTION DOES NOT EXIST.

****

Fig. 4(b)

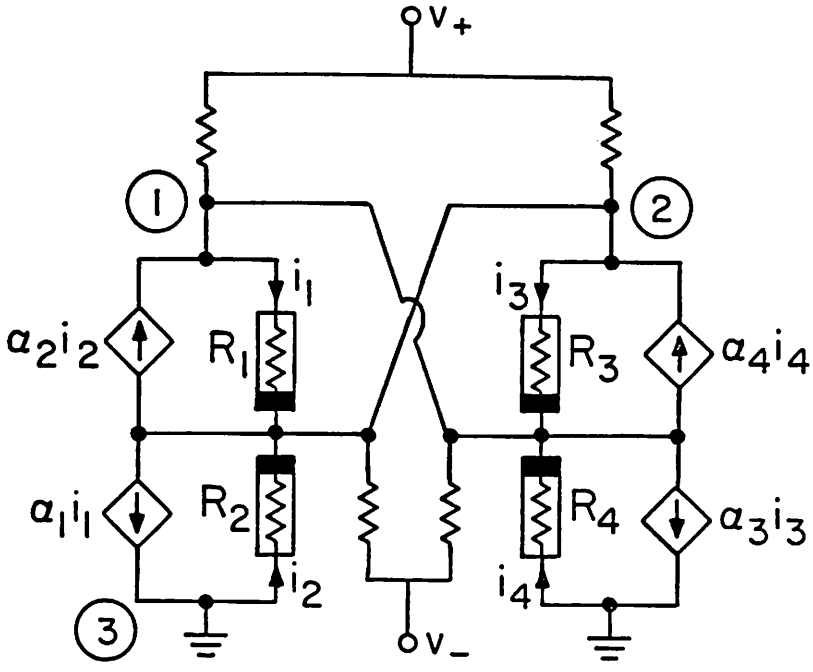

v+
1
2
i1
i3
α2i2
R1
R3
α4i4
α1i1
R2
R4
α3i3
i2
i4
3
v-

Fig. 5

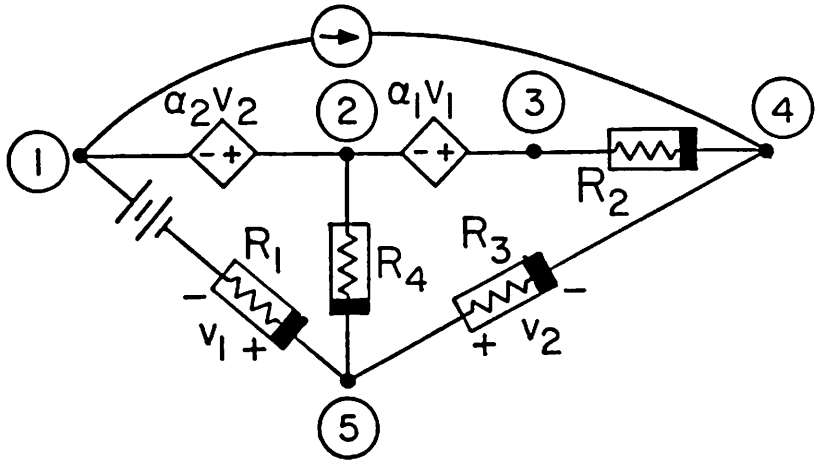

α2v2
2
α1v1
3
4
1
R2
R1
R4
R3
v1
v2
5

Fig. 7

UNIQF 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: Example 4(a)

```
R1    0   2
R2    0   3
R3    7   3
R4    5   3
R5    8   2
R6    6   2
R7    3   4
R8    2   4
V1    1   4
V2    0   1
F1    3   2    1   5  0.5005
F2    2   3    1   6  0.5005
F3    3   1    2   7  0.4
F4    2   1    3   8  0.4
```

STATES OF CIRCUIT ELEMENTS

Resistors	State
R1	open
R2	open
R3	open
R4	short
R5	open
R6	short
R7	open
R8	open

CCCS's	State	Max. Coeff.	Input Branch	Output Branch
F1	intact	0.5005	0 -->-- 2	2 -->-- 1
F2	intact	0.5005	0 -->-- 1	1 -->-- 2
F3	zeroed			
F4	zeroed			

determinant test: det(I+AG0) = -0.001

****

negative determinant found

SOLUTION IS NOT UNIQUE

****

Fig. 6(a)

UNIQF 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: Example 4(b)

```
R1    0   2
R2    0   3
R3    7   3
R4    5   3
R5    8   2
R6    6   2
R7    3   4
R8    2   4
V1    1   4
V2    0   1
F1    3   2    1   5  0.4995
F2    2   3    1   6  0.4995
F3    3   1    2   7  0.4
F4    2   1    3   8  0.4
```

****

negative determinant not found

UNIQUE SOLUTION EXISTS

****

Fig. 6(b)

```
UNIQF  1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input:  Example 5(a)

        R1    2   0
        R2    3   0
        R3    0   4
        R4    5   4
        I1    1   4
        V1    1   2
        E1    5   3    0   2  1.001
        E2    3   1    0   4  100

--------------------------------------------------------------------------------

STATES OF CIRCUIT ELEMENTS

Resistors     State
 R1            open
 R2            open
 R3            short
 R4            short

VCVS's        State       Max. Coeff.    Input Branch    Output Branch
 E1            intact       1.001          1 -->-- 0       1 -->-- 0
 E2            zeroed

determinant test: det(I+AG0) = -0.001

--------------------------------------------------------------------------------

                  ****************************************

                        negative determinant found

                          SOLUTION IS NOT UNIQUE

                  ****************************************
```

Fig. 8(a)

UNIQF 1.1

INSITE, Nonlinear Systems Group, U. C. Berkeley

Input: Example 5(b)

```
R1    2   0
R2    3   0
R3    0   4
R4    5   4
I1    1   4
V1    1   2
E1    5   3    0   2  0.999
E2    3   1    0   4  100
```

****

negative determinant not found

UNIQUE SOLUTION EXISTS

****

Fig. 8(b)

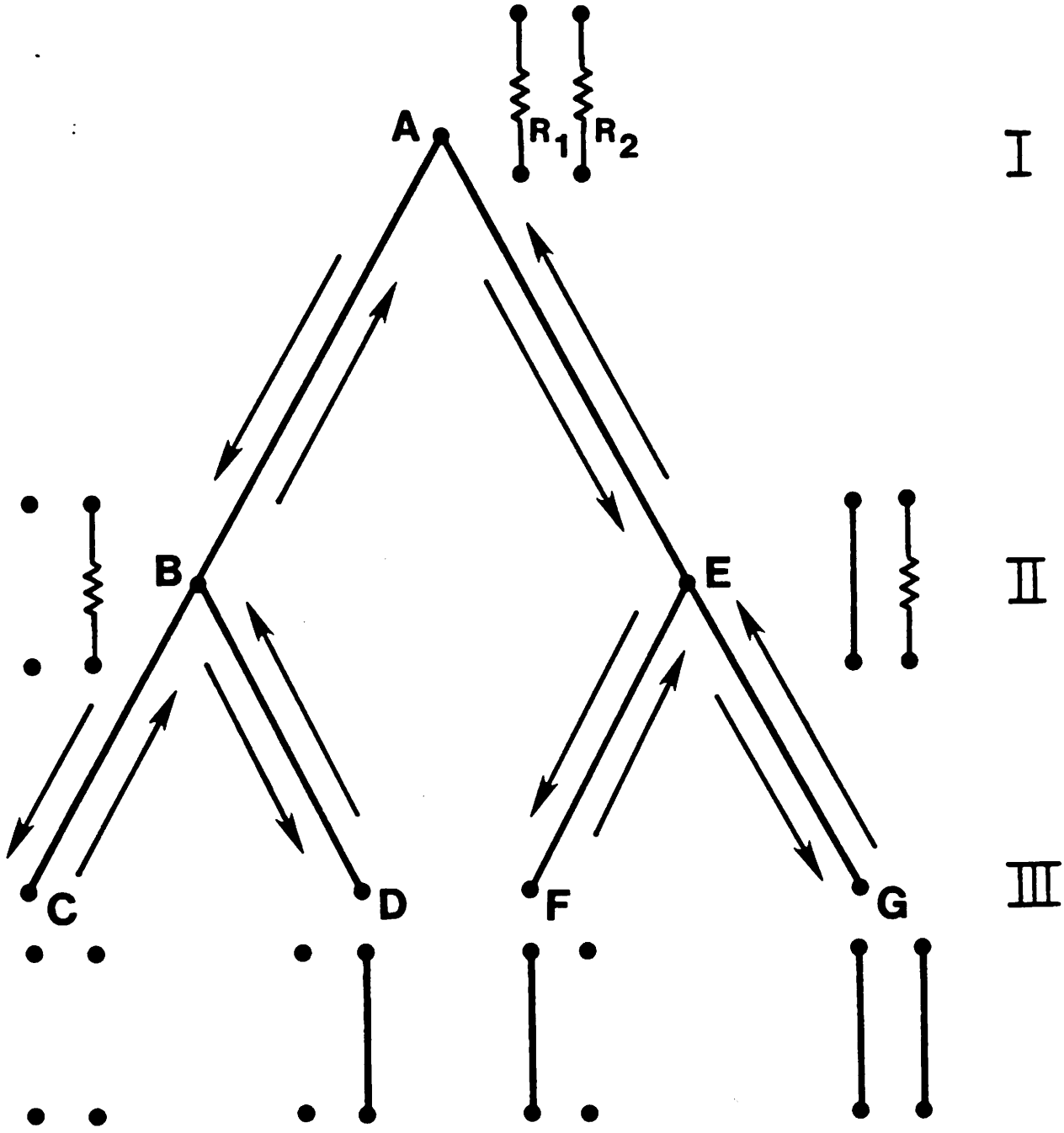


Fig. 9