線形回路の言語的表現とその記号解析への応用

菅原一孔† 松本康宏†† 小西亮介†

本稿では，回路および解析処理内容を言語的な表現方法で記述する一手法と，提案する記述法で表現された回路を汎用の数式処理言語系を用いて記号解析するシステムの概要について述べる．提案する回路記述手法では，回路全体を幾つかの部分回路の相互接続で表現する．そして全体の回路および部分回路を，それぞれ手続き型プログラミング言語におけるメインルーチンと関数のように取り扱う．この際，部分回路の再帰的な呼出しや部分回路間で回路変数などのデータの授受が行える．また回路中の素子名や節点番号の管理も解析システムが行う．提案した回路記述手法によって表現された回路の記号解析を，汎用の数式処理言語を利用して行うシステムを開発した．本解析システムでは，まず回路記述データを読み込み，解析するための汎用数式処理言語プログラムを生成する．次にこれを汎用数式処理言語系に読み込み，記号解析を行う．開発したシステムは提案する記述手法の特徴を生かし，しかも種々の汎用数式処理言語系に対応できるよう，基本的な構成にコンパイラ・インタプリタ方式を採用している．なお，ここではこのような汎用数式処理言語系のうち Mathematica をその例として用いている．回路の構成要素の記述方法や解析処理内容の記述方法について，複数の記述例を用いて説明している．また2つの回路解析例を用いて，回路の記述方法および記号解析結果を示し，本手法の有効性を確認している．

Circuit Description Language and Its Application to Symbolic Network Analysis

KAZUNORI SUGAHARA,† YASUHIRO MATSUMOTO†† and RYOSUKE KONISHI†

In this paper, an approach to describe electrical circuits as a linguistic form is discussed and a system which translates a circuit expression from the proposed form to a symbolic and algebraic manipulation program (SAMP) and analyzes a circuit in a symbolic form is proposed. In our circuit description form, the circuits to be analyzed are described as an interconnection of one main circuit and some sub-circuits. And they are treated as a main routine and sub-routines of ordinal procedural languages, respectively. In each of main and sub-circuits, names of circuit elements and node numbers are managed by the analyzing system independently. Because the recursive calling of sub-circuits are allowed, circuits which have regular structures like ladder-filters can be described efficiently. In our circuit analyzing system, a circuit described in the proposed linguistic form is translated to a SAMP and it is analyzed as a symbolic form following a resultant program. In this paper, Mathematica is used as an example of such a SAMP. However, proposed system can be easily modified according to the SAMP. The manner to describe circuit elements and analyzing processes in proposed form are explained by using some examples. And two circuit analyzing examples are also included to illustrate the efficiency of the proposed method.

1. はじめに

先に我々は汎用の数式処理言語系を積極的に利用して線形回路の記号解析を行う手法について報告した[1]．そこではまず，解析対象の回路の記述データを読み込み，回路の記号解析を行うための汎用数式処理言語プログラムを生成し，これを汎用数式処理言語系に読み込むことで解析を行った．この手法によると複素周波数 s や回路素子の値が記号の形で含まれた記号回路関数 (Symbolic Network Function, 以下 SNF) を求めることができた．このような汎用数式処理言語系を用いた回路の記号解析手法は，従来から知られていたパラメータ抽出法や，木アドミタンス積による方法，シグナルフローグラフによる方法[2)~5)]などとは異なり，数値的な繰り返し演算を必要としないため，それによる演算誤差が発生しないなどの特徴を有していた．

文献1)における回路の接続状況や解析処理内容を

† 鳥取大学工学部電気電子工学科
Faculty of Engineering, Tottori University
†† 神戸市立工業高等専門学校電子工学科
Department of Electronics, Kobe City College of Technology

与える入力データファイルの記述方法は，回路解析プログラム Spice[6]などに採用されているものであり，素子が接続されている節点番号と素子値などを羅列してゆくもので，簡便なため分かりやすいものであった．しかし，この記述方法では

- 等価回路への展開などはあらかじめ利用者が済ましておかなければならず，これに手間がかかるだけでなく，その記述が複雑になるために記述の誤りが生じやすい．
- あらかじめシステムで用意された素子や解析処理内容しか利用できず利用者の自由度が小さい．

などの問題があった．また，Spice のような数値的な回路解析プログラムと比較すると，Spice などは数値解を求めるだけなので個々の素子を管理する必要は特になく，回路の記述も簡便なものですむのに対し，記号解析を行う場合は，1つ1つの素子についてその素子名，素子値などを管理しなければならず，おのずと記述も複雑になってしまう．またどのような回路解析を行うのかその手順や，一部またはすべての素子について数値的な解析が必要な場合にも対応できるよう，各素子の数値的な素子値なども効率よく記述できることが望まれる．

そこで本稿では，このような素子の管理を効率よく行い，同時に回路の接続状況やその解析方法の記述を簡便に行うことができるようにするために，

- 手続型プログラミング言語のように「繰り返し」，「条件判断」などの制御構造および次項で述べる「関数・手続き」を導入することで，より効果的な回路記述を行えるようにする．
- 「関数・手続き」の考え方を導入することにより，例えばトランジスタなど，複数の素子でその等価回路が記述される素子が含まれる回路や，全体が幾つかの部分回路の相互接続により階層的に表現された回路の記述に際し，端子や素子の管理に関する諸問題を解消する．
- 利用者が新しい素子や解析処理内容を記述できるようにする．

ことなどを目的として，入力データを単なる接続情報の羅列としてではなく，回路および解析処理内容を記述する一種の言語（以下 SELⅡ 言語）として扱う手法について提案する．

SELⅡ 言語の文法は表記方法が比較的簡便でかつ最近特に利用の多い言語であるC言語に準ずるものになるよう工夫した．また，これを汎用数式処理言語系を利用した回路の記号解析システム（以下 SELⅡ システム）に応用した結果についても報告する．基本的な SELⅡ システムの処理の流れは文献1)と同様に，SELⅡ 言語に従って記述された回路記述ファイルを入力とし，汎用の数式処理言語プログラムを生成するプログラムジェネレータの形式をとっている．SELⅡ システムでは，利用する汎用の数式処理言語系の言語仕様に依存する処理とそうでないものを分離することにより，様々な汎用の数式処理言語系に対応した解析システムを構築することができる．本稿では汎用の数式処理言語系のうち比較的利用の多い Mathematica を採用した場合について述べる．

SELⅡ システムは抵抗，キャパシタ，インダクタ，相互結合インダクタ，独立電圧・電流源および電圧制御電流源（VCCS）を基本素子とし，それらとそれらの組み合わせで等価回路が表現できる素子からなる，定常状態にある線形回路の記号解析を行うプログラムである．また複素周波数 s を含む任意の素子の素子値を記号として扱い，回路中任意の枝の電流，電圧を求めることができる．なお本稿ではカットセット解析により回路解析を行う手法について述べるが，その他の解析手法の場合でも同様な解析を行うことができる．

本稿では，まず提案する回路の記述方法について2章で述べ，3章ではこれを回路の記号解析に応用するにあたり必要な事項について述べる．また4章では本解析プログラムによる解析例を示し，本手法の有効性を確認する．

2. 回路の記述方法について

提案する回路記述手法の完全な書式を拡張バッカス・ナウア記法[7]によって表現したものを付録に示す．そのうち，各種の操作を行う命令文についてその処理内容を**表1**にまとめる．また後で述べる前処理用の擬似命令を**表2**に示す．回路の接続状況を記述する際や行う解析手続きを記述する際には，これらと付録中(c)の制御文やその他各種の文を組み合わせて効率的に記述する．なお，表2の処理内容の説明に出てくるシステム変数の一覧とその意味を**表3**にまとめる．

2.1 回路記述ファイルの書式

付録中(a)に示されているとおり回路記述ファイル（プログラム）は回路の接続状態を示す circuit 部と，解析処理内容を示す control 部からなる．

circuit 部はおもに回路の接続状況を記述する部分であり，先に述べたようにC言語に似た記述方法を

表 1 命令文一覧
Table 1 Commands.

操作文の種類	文	処理内容
行列操作	putX(x, y, sign)	接続行列Xのx行y列の要素として符号定数 sign を置く.
	putYb(x, y, str)	アドミタンス行列 Yb のx行y列の要素として str を置く.
	putVt(x, str)	木枝電圧ベクトル Vt のx番目の要素として str を置く.
	putIs(x, str)	電流ベクトル Is のx番目の要素として str を置く.
文字列操作	copy(dest, src)	dest の文字列型変数に src の文字列をコピーする.
	add(dest, src)	dest の文字列の後に src の文字列を結合する.
数式言語操作	math(p1, p2, ..., pn)	数式処理言語に対する命令文として p1, p2, ..., pn を数式処理言語プログラムファイルにそのまま書き出す.
	outrf(p1, p2, ..., pn)	数式処理言語に対する命令文として p1, p2, ..., pn を数式処理言語プログラムファイルにそのまま書き出す. ただし math 文とことなり, 書き出した命令についての応答を出力先へ出力するための指示が自動的に付加される.
	error(p1, p2, ..., pn)	回路記述内で矛盾が発生した場合に, 回路記述を中断・強制終了させるための命令. エラーメッセージとして p1, p2, ..., pn を標準出力に書き出した後, 解析インタプリタが停止する.
	file(str)	結果出力先のファイル名を str で示されるファイル名のファイルに変更する.
	setconst(id)	文字列定数 NAME の参照先として id を設定する.
	asgconst(n)	素子番号nの素子の素子値を次回の outrf 命令実行時に一時代入する. 一度 outrf が実行されると, 代入指定はリセットされる.
その他	lenof(str)	文字列 str の文字列長を与える.
	strtoint(str)	文字列 str を数値に変換する.
	inttostr(n)	整数型データnの値を文字列に変換する.
	getc(str, n)	文字列 str の第n文字目を取得する.

表 2 前処理用擬似命令一覧
Table 2 Pseudo instructions for the pre-processing.

命令	処理内容
#include file	file の内容をその場所に展開する.
#indepelm	回路中の独立な素子の数を表すシステム変数 INDEPC を1増加させる. 素子のうち, 独立素子についてのみ, この擬似命令を素子記述の先頭に置く.
#depelm	システム変数 DEPC を1増加させる. 素子のうち, #indepelm を置かなかったものに, この擬似命令を置く.
#vsrc	独立電圧源について, これを置く. 展開後は, 回路中の独立電圧源の数を表すシステム変数 VSRCC を1増加させる.
#csrc	独立電流源について, これを置く. 展開後は回路中の独立電流源の数を表すシステム変数 CSRCC を1増加させる.

表 3 システム変数一覧
Table 3 System variables.

命令	処理内容
INDEPC	回路中の独立な（接続行列で列を占有する）素子の数を表す.
DEPC	縦続な（接続行列で列を占有しない）素子の数を表す.
TERMC	回路中の端子の数を表す. 接続行列の行数に対応する.
VSRCC	回路中の独立電圧源の数を表す.
CSRCC	回路中の独立電流源の数を表す.

とっている. これは, 1つの回路全体を表す部分（以下 main 回路）と, 0個以上の部分回路を表す部分（以下部分回路）から構成される. ここで main 回路や部分回路内ではそれぞれ独立して素子名を利用することができるだけでなく, 節点番号も独立して設定することができる. これによって回路に同じ部分回路が複数含まれる場合でも, その部分回路を示す同一の部分回路をその回数呼び出すだけで回路を表現することができる. また, 部分回路では再帰的な呼出しを許しており, 梯子型フィルタ回路などのように規則的な構造を持つ回路を効率的に表現することができる.

```
R(t1, t2) {s1}
{
    string str;
#indepelm
    setconst(s1);
    putX(t1, INDEPC, PLUS);
    putX(t2, INDEPC, MINUS);
    copy(str, "1/(");
    add(str, NAME);
    add(str, ")");
    putYb(INDEPC, INDEPC, str);
}
```

図 1 抵抗の記述方法
Fig. 1 The description of a resister.

```
E(t1, t2) {s1}
{
    string str;
#indepelm
#vsrc
    setconst(s1);
    putX(t2, INDEPC, PLUS);
    putX(t1, INDEPC, MINUS);
    putVt(INDEPC, NAME);
    putIs(INDEPC, "*");
}
```

図 2 独立電圧源の記述方法
Fig. 2 The description of an independent voltage source.

control 部は circuit 部で記述された回路について，どのような解析を行うのか，その手順を記述する部分である．ここでも表1中の諸命令と付録中の各種の文を利用することができる．

以下，circuit 部，および control 部の書式について述べる．

2.2 circuit 部

回路中で利用される代表的な素子，例えば抵抗は表1の命令を使って**図1**のようにC言語に似た表記方法で記述される．

図1において，前処理用の擬似命令 #indepelm は，この素子が節点を有する素子であることを示しており，プログラム内部では現在までに入力された素子の数を表すシステム変数 INDEPC を1増加させる処理をする．このシステム変数 INDEPC は，その素子の全回路中での通し番号に対応している．相互インダクタなどそれ自身が回路中で端子を持たない素子の場合や部分回路の場合は，この擬似命令は使わない．符号定数 PLUS, MINUS はそれぞれ +1, −1 を表す．また，t1, t2 はこの抵抗が接続される2つの節点番号を表す．また s1 は文字列型変数であり数値的な素子値を指定する必要がある場合などに用いる．

3章で述べるように回路を解析するためには，その回路の接続状況を表す接続行列 X や，素子アドミタンス行列 Y_b あるいは木の枝電圧ベクトル $\boldsymbol{V}_t$，および電流源ベクトル $\boldsymbol{I}_s$ 等が必要となる．今の場合，putX(t1, INDEPC, PLUS) 命令で接続行列 X の t1 行 INDEPC 列に +1 が，また putX(t2, INDEPC, MINUS) 命令で t2 行 INDEPC 列に −1 が代入される．これによって INDEPC の素子番号をもつ抵抗が端子 t1-t2 間に接続されており，そこに流れる電流の向きを表す基準方向は t1 から t2 に向いていることが表現される．同様に putYb(INDEPC, INDEPC, str) 命令で，アドミタンス行列 Y_b の INDEPC 行 INDEPC 列に文字列 str が代入される．抵抗の場合，この命令の前の文字列操作命令 copy 命令と add 命令によって，文字列 str にはこの抵抗の素子名（NAME という変数に格納されている）を用いて "(1/NAME)" という文字列が設定されている．以上の操作によって抵抗に関して，カットセット解析に必要な行列生成の処理は終了する．

独立電圧源の場合は**図2**のように記述される．この場合も抵抗の場合と同じように接続行列 X，木の枝電圧ベクトル $\boldsymbol{V}_t$ および電流源ベクトル $\boldsymbol{I}_s$ を生成している．

回路記述に際して，変数としては付録中(b)の部分に示しているとおり，文字列型 (string 型) 変数と整数型 (int 型) 変数を宣言することができ，それぞれ付録のうちで(d)の部分の文字列データと数値データに対応する．ここで文字列型変数は素子名を表す文字列を操作する際などに，整数型変数は繰り返し回数などの変数を宣言する際などに利用する．特に回路中の節点番号は，対応する数値データで数値の前にピリオドをつけて表すことで，それ以外の数値と扱いを区別している．

このような回路の構成要素は表1の各種命令と付録中の各種の制御文を用いて表現すればよい．またそれらのうちで基本的な素子については，その特性を共通の標準素子定義ファイル stdelm.h に記述しておき，これを読み込むことで回路を簡便に記述することができる．なお，以下ではこのように標準素子定義ファイルに記述されたものをマクロと呼ぶことにする．現在，stdelm.h にはここで述べた抵抗や独立電圧源以外に，トランジスタなどの能動素子を含め 10 種類の素子がマクロとして記述されている．

2.3 control 部

表1に示されている出力命令を利用してどのような解析処理を行うか，その手順を記述する部分である．

```
GetV(where,cns){}
{
    int i;

    if (cns!=0) {
        for (i=1;i<=INDEPC;i=i+1) asgconst(i);
        for (i=1;i<=DEPC;i=i+1) asgconst(i+255);
    }
    outrf("Vb[[",where,"]]");
}
```

図 3 枝電圧を求める場合の記述方法
Fig. 3 The expression to get a branch voltage.

```
GetAbsV(where,from,to){}
{
    int i;

    for (i=1;i<=INDEPC;i=i+1) asgconst(i);
    for (i=1;i<=DEPC;i=i+1) asgconst(i+255);
    math("s = I w");
    outrf("tmpexp = Vb[[",where,"]] ");
    outrf("Plot[Abs[tmpexp],{w,",from,",",to,"}]");
}
```

図 4 枝電圧の振幅特性を求める場合の記述方法
Fig. 4 The expression to get frequency characteristics of a branch voltage.

周波数特性やある素子に関する素子感度，その他各種の特性についても，それを求める手順を表1中の制御文を利用して記述することができる．例えば枝の電圧を式の形で求めたり，その振幅特性図を求める場合，それぞれ図3，4のように記述すればよい．

なお，この場合も circuit 部と同様に代表的な解析手順をマクロとして標準解析定義ファイル analysis.h に記述しておき，これを読み込むことで解析手順を簡便に表現することができる．

3. 数式処理言語による回路記号解析

3.1 回路記号解析処理の流れ

SELⅡ システムはC言語と awk 言語により記述されたプログラム群であり，回路の解析を行うための汎用の数式処理言語プログラムを生成する，いわゆるプログラムジェネレータの形式をとっている．生成されたプログラムは汎用の数式処理言語系に受け渡され，その上で実際の解析処理は行われることになる．

図5に SELⅡ システムの処理の流れを示す．以下では図5の流れにそって説明を加えて行く．

図5に示されているように，SELⅡ システムは基本的にはコンパイラ・インタプリタ方式[8]を採用しているが，様々な言語仕様の汎用数式処理言語系に対応できるよう，その言語仕様に依存しない処理を行う解析コンパイラおよび解析インタプリタと，これに依存する処理を行う前処理部と数式処理言語プログラム生成部から構成されている．図中，2重枠で書かれている部分は，利用する汎用数式処理言語系の言語仕様に依存しない部分を表す．

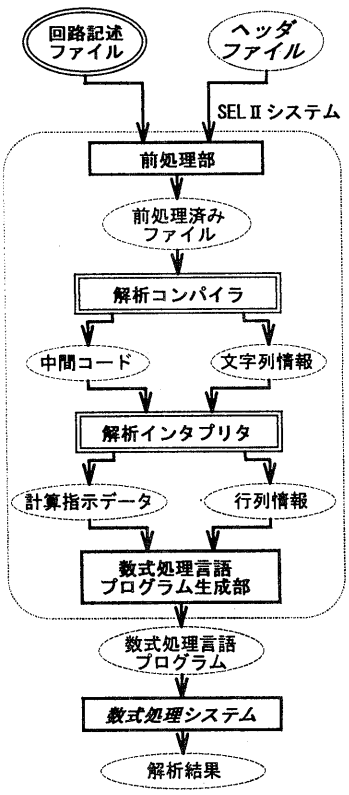


図 5 回路解析の流れ
Fig. 5 The flow chart of the circuit analysis.

まず，前処理の部分では先に述べた回路記述ファイルおよびその中に擬似命令 #include によって指定されている標準素子定義ファイルや標準解析定義ファイルなどを読み込む．

次に解析コンパイラでは，文法的な誤りを確認しながら，回路解析に必要な各種行列を作るための手順や素子名対応表を作るための手順を示す中間コードと文字列情報を生成する．解析インタプリタではこの中間コードに従い，文字列情報を利用して必要な行列等を生成する．この作業も使用する汎用数式処理言語系には依存しないのでこれを計算指示データと行列情報としてファイルに出力する．

続く数式処理プログラム生成部では，計算指示データと行列情報をもとに，利用する汎用数式処理言語系

に対応した回路解析プログラムを生成する．先に述べたように，このような構成をとることで，異なった言語仕様の汎用数式処理言語系を用いる場合でも，標準解析定義ファイルと前処理部および数式処理プログラム生成部だけそれに合せて用意するだけですみ，中心的な役割を果たす解析コンパイラや解析インタプリタの部分は共通のものを利用できる．

3.2 回路の記号解析方法

回路の記号解析はカットセット解析により行うものとした．これは

$$Q\cdot Y_b\cdot Q^t\cdot V_t=-Q\cdot I_S \qquad (1)$$

で表されるカットセット方程式を解くことに相当する．ここで，Q は回路の基本カットセット行列，Y_b は素子アドミタンス行列，V_t は木の枝電圧，I_S は電流源ベクトルである．

式(1)の $Q\cdot Y_b\cdot Q^t$ と $-Q\cdot I_S$ を計算するにあたり，回路に電圧源が含まれている場合，V_t と I_S にそれぞれ未知変数である枝電圧 v_t と独立電圧源の枝電流 I_E が含まれる．従ってこのままでは v_t と I_E について解くことができないので，式(1)を

$$\begin{bmatrix}M_{11} & M_{12}\\ M_{21} & M_{22}\end{bmatrix}\begin{bmatrix}E\\ v_t\end{bmatrix}=\begin{bmatrix}-I_E+N_1\cdot J\\ N_2\cdot J\end{bmatrix} \qquad (2)$$

のように既知の部分行列 N_1, N_2, M_{11}, M_{12}, M_{21}, M_{22}, E, J と未知の v_t, I_E に分解する．ただしここでは，木枝電圧 V_t を独立電圧源電圧 E と未知の木枝電圧 v_t を用いて $[E\ v_t]^t$，また回路の接続状態によって定まる未知の変数を含まないベクトル $\boldsymbol{N}_1$, $\boldsymbol{N}_2$, $\boldsymbol{M}_{12}$, $\boldsymbol{M}_{12}$, $\boldsymbol{M}_{21}$, $\boldsymbol{M}_{22}$ と既知の独立電流源電流 J および未知の独立電圧源の電流 I_E を用いて，式(1)における $Q\cdot Y_b\cdot Q^t$, $-Q\cdot I_S$ を

$$Q\cdot Y_b\cdot Q^t=\begin{bmatrix}\boldsymbol{M}_{11} & \boldsymbol{M}_{12}\\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22}\end{bmatrix} \qquad (3)$$

$$-Q\cdot I_S=[-I_E+\boldsymbol{N}_1\cdot J\ \ \boldsymbol{N}_2\cdot J]^t$$

と書いた．ここで独立電圧源の枝の数を n，独立電圧源以外の木の枝数を m とすると $\boldsymbol{M}_{11}$, $\boldsymbol{M}_{12}$, $\boldsymbol{M}_{21}$, $\boldsymbol{M}_{22}$ はそれぞれ $n\times n$, $n\times m$, $m\times n$, $m\times m$ の大きさを持ち，また $\boldsymbol{N}_1$, $\boldsymbol{N}_2$ は $n\times 1$, $m\times 1$ の大きさのベクトルとなる．

結局，式(1)のカットセット方程式の解，v_t と I_E はそれぞれ

$$v_t=\boldsymbol{M}_{22}^{-1}\cdot(\boldsymbol{N}_2\cdot J-\boldsymbol{M}_{21}\cdot E) \qquad (4)$$

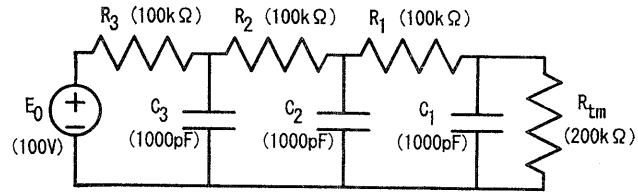
$$I_E=-\boldsymbol{M}_{11}\cdot E-\boldsymbol{M}_{12}\cdot v_t+\boldsymbol{N}_1\cdot J$$

で与えられる．さらに得られた解からすべての枝の電流，電圧は求めることができる[9]．式(4)の $\boldsymbol{M}_{ij}$ $(i,j=1,2)$, $\boldsymbol{N}_i$ $(i=1,2)$, E, J を解析コンパイラ，解析インタプリタによって生成し，これを利用する汎用数式処理言語系の言語仕様で表現したものを汎用数式処理言語系に受け渡し，そこで式(4)を式の形で計算することにより解を得る．また求めた枝電流，枝電圧から回路の入出力の伝達関数や極・零点の位置，係数感度なども式の形で求めることができる．

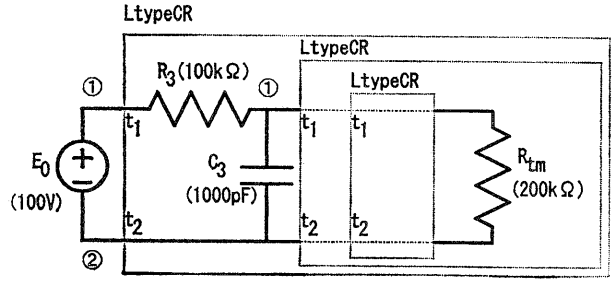
さらに，得られた結果に素子値や周波数の値などを代入することにより，様々な形式の数値解や特性図を求めることができる．この場合の数値解は素子の値を表す変数に関して陽の形で表された SNF に単に値を代入するだけであるため，繰り返し計算によって求められた従来の数値解析の解に比べ，精度のよいものを得ることが期待できる．

4. 解 析 例

3章で述べたプログラムによる回路の記述例およびその記号解析結果を示す．ただし，ここでは回路の記述手法と解析手順を具体的に示すことを目的とするた



(a) 3段 R-C フィルタ回路
(a) The R-C ladder filter.



(b) 部分回路の再帰的呼び出しによる表現
(b) The expression of the R-C ladder filter by using the recursive calling of sub-circuits.

図 6 解析例 1
Fig. 6 Example 1.

めに，本手法を用いるまでもないような簡単な回路，あるいはすでに解が知られているものについての解析を行う．

4.1 解析例1 （C-R回路：再帰的に部分回路を呼び出す例）

図6（a）に示される回路の記号解析を行う．

この例では3段の C-R 梯子型回路に終端抵抗が接続されているものであり，C-R梯子型回路の1段ごとの部分回路の再帰的な呼び出しにより表現されている．この考え方を示したのが図6（b）である．この解析例の回路記述ファイルは**図7**のようになる．

ここで，E[0](.2,.1){"100"} は節点1-2間に独立電圧源 E_0 が接続されており，数値解を求める際にはその電圧源の電圧の標準は 100[V] であることを示している．なお，数値の前にピリオドを打って節点番号を表しているのは先に述べたとおりである．これらの電圧源や抵抗は先に述べたように，標準素子定義ファイル stdelm.h にその行列上での表現方法がマクロとして記述されているものとした．

```
#include stdelm.h
#include analysis.h

LtypeCR(t1,t2,n){}
{
    term 1;

    if (n>1) {
        R[@n](t1,.1){"100%k"};
        C[@n](.1,t2){"1000%p"};
        n=n-1;
        LtypeCR_(.1,t2,n){};
    } else {
        R[@n](t1,.1){"100%k"};
        C[@n](.1,t2){"1000%p"};
        R[tm](.1,t2){"200%k"};
    }
}

main(n){}
{
    term 2;

    E[0](.2,.1){"100"};
    LtypeCR_(.1,.2,n){};
}

control(){}
{
    main_(3){};
    GetV_(¥R[tm],0){};
    GetAbsV_(¥R[tm],0,100000){};
    GetArgV_(¥R[tm],0,100000){};
}
```

図7 図6の回路の記述
Fig. 7 The description of Fig. 6.

このリストの中で部分回路（LtypeCR 部分回路）を表現しているところで LtypeCR 部分回路自身が再帰的に呼び出されていることが分かる．

この例では control 部からまず main 回路内に LtypeCR 部分回路が3段接続されているとして呼び出し，終端抵抗にかかる電圧を求め，その電圧の振幅特性と位相特性を求めている．得られた結果のうち終端抵抗にかかる電圧を**図8**に示す．

さらにこの解析例の場合，同じ回路構造で3段以外のフィルタの場合でも，main 回路の呼出回数を変更するだけでよい．この点，文献1)の回路記述手法では，あらためて節点番号をふりなおし，回路の記述をしなおさなければならなかったことと比較すると，ここで提案する記述手法の有効性が確認できる．

4.2 解析例2 （部分回路の相互接続の例）

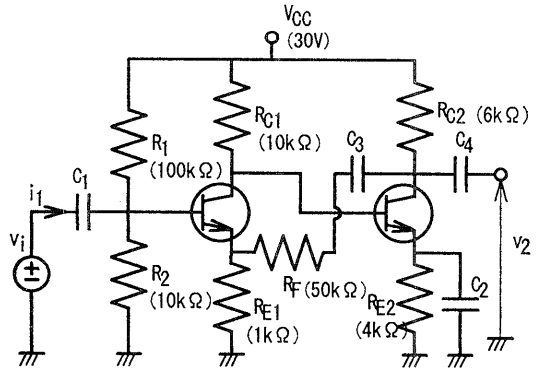
幾つかの部分回路の相互接続で表現する場合の例として，**図9**（a）に示されるトランジスタ2段増幅回路を考える[10]．この回路の低周波数領域における交流等価回路は同図（b）となる．

解析の際にはトランジスタの等価回路として同図（c）に示される等価回路を用いた．さらに，この等価回路中に存在する電流制御電流源（以下 CCCS）は同図（d）の等価回路で表現することとした．この回路の回路記述ファイルを**図10**に示す．この場合，main 回路から，トランジスタを表す Tr 部分回路が呼び出され，さらにその部分回路から CCCS を表現している CCCS 部分回路が呼び出されていることが分かる．

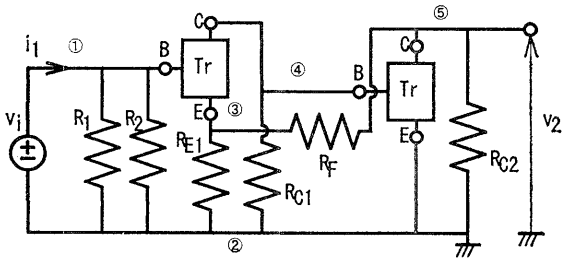
この回路で信号源 v_i から流れ出る電流 i_1 を求めると，**図11**のようになる．なお，同図（a）には v_i のみ記号としてとらえた場合の解を，同図（b）には v_i と抵抗 R_F を記号とした場合の解を示している．このように SELII 言語では，回路中の素子値ごとにそれを記号として扱うのか，数値として扱うのかを選択しながら解析処理を行うことができる．

```
(E0*Rtm)/(Rtm + R1 + R2 + R3 + C1*Rtm*R1*s + C1*Rtm*R2*s + C2*Rtm*R2*s +
    C2*R1*R2*s + C1*Rtm*R3*s + C2*Rtm*R3*s + C3*Rtm*R3*s + C2*R1*R3*s +
    C3*R1*R3*s + C3*R2*R3*s + C1*C2*Rtm*R1*R2*s^2 + C1*C2*Rtm*R1*R3*s^2 +
    C1*C3*Rtm*R1*R3*s^2 + C1*C3*Rtm*R2*R3*s^2 + C2*C3*Rtm*R2*R3*s^2 +
    C2*C3*R1*R2*R3*s^2 + C1*C2*C3*Rtm*R1*R2*R3*s^3)
```

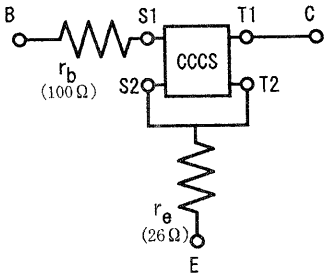
図8 解析結果
Fig. 8 The result.



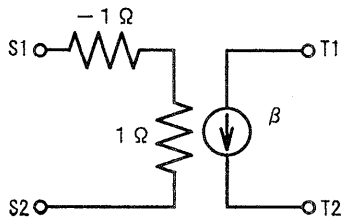
(a) トランジスタ2段増幅回路
(a) The transistor amplifier.



(b) (a)の低周波交流等価回路
(b) The equivalent circuit of (a).



(c) トランジスタの等価回路 (Tr)
(c) The equivalent circuit of transistors.



(d) 電流制御電流源の等価回路 (CCCS)
(d) The equivalent circuit of current controlled current sources.

図 9 解析例 2
Fig. 9 Example 2.

```
#include stdelm.h
#include analysis.h

CCCS(t1,t2,s1,s2){ratio}
{
    term 1;
    int RPcount;

    R[p](s1,.1){"1"};
    RPcount=INDEPC;
    R[m](.1,s2){"-1"};
    VCCS[Q](t2,t1,RPcount){ratio};
}

Tr(TB,TC,TE){}
{
    term 2;

    R[b](TB,.1){"100"};
    CCCS[Q](.2,TC,.1,.2){"99"};
    R[e](.2,TE){"26"};
}

main(){}
{
    term 5;

    E[i](.2,.1){"vi"};
    R[1](.1,.2){"100%k"};
    R[2](.1,.2){"10%k"};
    Tr[1](.1,.4,.3){};
    R[E1](.3,.2){"1%k"};
    R[F](.5,.3){"50%k"};
    Tr[2](.4,.5,.2){};
    R[C1](.4,.2){"10%k"};
    R[C2](.5,.2){"6%k"};
}

control(){}
{
    main_(){};
    GetI_(¥E[i],1){};
    GetIC_(¥E[i],¥R[F]){};
}
```

図 10 図9の回路の記述
Fig. 10 The description of Fig. 9.

5. おわりに

本稿では回路を手続き型言語のメインルーチンのように取り扱い，これを手続型言語の関数と同じように表現される幾つかの部分回路の相互接続で表す一記述手法を提案した．本記述手法によると利用者は部分回路ごとに素子番号の管理などを行うだけでよく，回路全体については解析システムが自動的に管理する．

さらに，本手法では部分回路の再帰的な呼び出しを許したため，梯子型フィルタ回路などのような規則的な回路構造を持つ回路を効率良く表現することができた．

本記述手法を代表的な汎用数式処理言語系の一つである Mathematica を利用して，定常状態にある線形

```
(-734486983*vi)/6611345300000
```

(a) 入力電圧 v_i を記号として得た入力電流 i_1 の解析結果
(a) The result of input current i_1 considering the input voltage v_i as a symbolic parameter.

```
-((656401033000 + 1561719*RF)*vi)/(100000*(59592003000 + 130429*RF))
```

(b) 入力電圧 v_i と抵抗 RF を記号として得た入力電流 i_1 の解析結果
(b) The result of input current i_1 considering the input voltage v_i and the resister R_F as symbolic parameters.

図 11 回路解析結果 2
Fig. 11 Results.

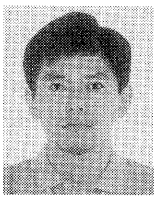
回路の記号解析を行うシステムを開発し，各種の回路を解析した．その結果，正しい解析結果を得たと共に，ここで提案した回路の記述手法の有用性を確認した．

今後は，回路の過渡解析や分布定数回路の解析などに，本稿で提案した回路の記述手法や汎用数式処理言語系を用いた解析手法を応用することが残されている．

謝辞 本研究を進める上で貴重な御意見を頂きました東京工業大学西原明法助教授に深謝します．また貴重な御意見を頂きました査読者の方々に感謝します．なお，本研究の一部は，平成5年度文部省科学研究費助成金（奨励研究(A) 課題番号 05750374）による．

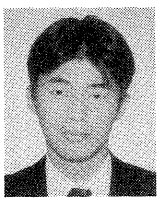
参 考 文 献

1) 菅原一孔，柿本秀樹，堂野正弘，落山謙三：数式処理プログラムによる線形回路の記号解析，情報処理学会論文誌，Vol. 30, No. 6, pp. 779-785 (1989).
2) 佐川雅彦：線形回路の記号解析，信学誌，Vol. 63, No. 5, pp. 468-474 (1980).
3) Alderson, G. E. and Lin, P. M.: Computer Generation of Symbolic Network Functions—A New Theory and Implementation, *IEEE Trans. Circuit Theory*, Vol. CT-20, No. 1, p. 48 (1973).
4) Lin, P. M.: A Survey of Application of Symbolic Network Functions, *IEEE Trans. Circuit Theory*, Vol. CT-20, No. 11, p. 732 (1973).
5) 佐川雅彦，北沢仁志：パラメータ抽出操作による線形回路の記号解析，信学論(A)，Vol. J60-A, No. 8, pp. 725-732 (1977).
6) Tuinenga, P. W.: *SPICE—A Guide to Circuit Simulation & Analysis Using PSpice*, Prentice Hall (1988).
7) Jensen, K. and Wirth, N. (原田賢一訳)：PASCAL, 培風館 (1988).
8) 佐々政幸：プログラミング言語処理系，岩波書店 (1989).
9) 小野田真穂樹：現代回路理論，昭晃堂 (1983).
10) 藤井信生：アナログ電子回路，昭晃堂 (1992).



菅原 一孔（正会員）

1956年生．1977年神戸市立工業高等専門学校電気工学科卒業．1979年山梨大学工学部電気工学科卒業．1981年東京工業大学大学院理工学研究科電子物理工学専攻修士課程修了．同年神戸市立工業高等専門学校電気工学科講師．同助教授，同校電子工学科助教授を経て，1994年鳥取大学工学部電気電子工学科助教授．その間回路理論，電子回路，ディジタル信号処理，数式処理言語の応用などの教育，研究に従事．工学博士．IEEE，電子情報通信学会各会員．



松本 康宏

1975年生．1990年神戸市立工業高等専門学校電子工学科入学．回路の記号解析や数式処理言語の応用に興味をもつ．平成5年度電気関係学会関西支部連合大会奨励賞受賞．電子情報通信学会会員．

小西 亮介

1946年生．1968年神戸大学工学部計測工学科卒業．1970年同大大学院修士課程修了．同年鳥取大学工学部電子工学科助手，1977年同助教授．1992年同教授となり現在に至る．その間，薄膜・表面物理に関する研究およびマイクロコンピュータの計測への応用に関する研究に従事．工学博士．電子情報通信学会，応用物理学会，計測自動制御学会，日本真空協会各会員．

付録 SELII の文法（コメント文は除く）

```
プログラム      = {circuit部} circuit部 control部 .
circuit部       = 関数名 関数ヘッダ "{" 関数本体 "}" .                    (a)
control部       = "control" 関数ヘッダ "{" 関数本体 "}" .
関数名          = 識別子 .
関数ヘッダ      = "(" [識別子並び] ")" "{" [識別子並び] "}" .
関数本体        = 端子数宣言 変数宣言 文並び .
端子数宣言      = ["term" 数値 ";"] .
変数宣言        = {( 整数型変数宣言 | 文字列型変数宣言 )} .
文字列型変数宣言= "string"識別子並び ","
整数型変数宣言  = "int" 識別子並び ","                                   (b)
文並び          = { 文 } .
文              = 制御文 | 操作文 | 関数呼び出し文 | 代入文 | 群文 |";" .
制御文          = if文 | if-else文 | while文 | for文 .                   (c)
操作文          = ( 行列操作文 | 文字列操作文 | 数式言語操作文 ) ";"
行列操作文      = putX文 | putYb文 | putVt文 | putls文 .
文字列操作文    = add文 | copy文 .
数式言語操作文  = math文 | outrf文 | error文 | file文 | setconst文 | asgconst文 .
関数呼び出し文  = 識別子 添え字 "("[整数データ並び]")"
                  "{" [文字列データ並び]"}"" " .
添え字          = 省略添え字 | 非省略添え字 .
省略添え字      = "_" .
非省略添え字    = "[" 添え字実体 "]" .
添え字実体      = "@" 識別子 | 純文字列 .
代入文          = 代入式 ";" .
代入式          = ( システム定数 | 識別子 )"=" 式 .
群文            = "{" 文並び "}" .
if文            = if接頭部 文 .
if-else文       = if接頭部 文 "else" 文 .
if接頭部        = "if""(" 条件式 ")" .
while文         = while接頭部 文 .
while接頭部     = "while""(" 条件式 ")" .
for文           = for接頭部 文 .
for接頭部       = "for""(" 代入式 ";" 条件式 ";" 代入文 ")" .
putX文          = "putX""(" 式 "," 式 "," 符号定数 ")" .
putYb文         = "putYb""(" 式 "," 式 "," 文字列データ ")"
putls文         = "putls""(" 式 "," 文字列データ ")" .
putVt文         = "putVt""(" 式 "," 文字列データ ")" .
add文           = "add""(" 識別子 "," 文字列データ ")" .
copy文          = "copy""(" 識別子 "," 文字列データ ")" .
math文          = "math""(" [パラメータ並び] ")" .
file文          = "file""(" 文字列 ")" .
outrf文         = "outrf""(" [パラメータ並び] ")" .
error文         = "error""(" [パラメータ並び] ")" .
asgconst文      = "asgconst""(" 式 ")" .
setconst文      = "setconst""(" 識別子 ")" .
```

```
識別子並び      = { 識別子 ",") 識別子 .
整数データ並び  = { 式 ",") 式 .
文字列データ並び= {( 文字列データ | 省略子 ) ",") ( 文字列データ | 省略子 ) .
パラメータ並び  = {( 文字列データ | 式 ) ",") ( 文字列データ | 式 ) .
条件式          = { 論理式 ( "&"|"|" ) } 論理式 .
論理式          = "(" 論理式 ")"|"!" 論理式 | 式 関係演算子 式 .
関係演算子      = "=="|"!="|"<"|"<="|">"|">=" .
式              = 単純式 { 加減演算子 単純式 } .
単純式          = 因子 { 乗除演算子 因子 } .
因子            = ["-"] 分子 |"(" 式 ")" .
分子            = 数値データ | lenof分子 | strtoint分子 | 素子名 .
乗除演算子      = "*"|"/" .
加減演算子      = "+"|"-" .
システム変数    = "INDEPC"|"DEPC"|"TERMC"|"VSRCC"|"CSRCC".
文字列データ    = 識別子 | 文字列 |"NAME"| inttostr分子 | getc分子 .
数値データ      = 識別子 | システム変数 | 数値 |"." 数値 .                    (d)
符号定数        = "PLUS"|"MINUS" .
lenof分子       = "lenof""(" 文字列データ ")".
strtoint分子    = "strtoint""(" 文字列データ ")" .
inttostr分子    = "inttostr""(" 式 ")".
getc分子        = "getc""(" 識別子 "," 式 ")".
素子名          = { 単位名 } .
単位名          = "¥" 識別子 非省略添え字 .
文字列          = """" 裸文字列 """" .
裸文字列        = {( 任意の文字 | SI 接頭語 )} .
純文字列        = {( 英字 | 数字 )} ( 英字 | 数字 ) .
SI 接頭語       = "%" ( "M"|"k"|"m"|"u"|"p" ) .
識別子          = 英字 {( 英字 | 数字 )} .
数値            = { 数字 } 数字 .
数字            = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9" .
英字            = "a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|
                  "m"|"n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|
                  "y"|"z"|"A"|"B"|"C"|"D"|"E"|"F"|"G"|"H"|"I"|"J"|
                  "K"|"L"|"M"|"N"|"O"|"P"|"Q"|"R"|"S"|"T"|"U"|"V"|
                  "W"|"X"|"Y"|"Z"
```

EBNF（拡張バッカス・ナウア記法）の凡例

メタ記号	意　味
=	定義
|	選択
.	生成規則の終わり
[X]	空またはX
{X}	0回以上のXの繰り返し
(X|Y)	グループ化，XもしくはY
"XY"	終端記号XY

(平成6年1月26日受付)
(平成6年7月14日採録)