記号回路解析に適したすべての木の生成方法†

菱沼千明††

Abstract

Some properties for the sets of trees in an undirected graph are given. An effective method for finding all possible trees is presented and it is expressed recursively. The labor to find a set of all trees is much reduced, since a graph is defined as a union of parallel branches. In order to treat topological properties on the computer, binary list structure is used to represent a graph. The method can be implemented effectively and esthetically by a computer language LISP.

The method is applicable to symbolic network analysis.

1. まえがき

グラフ理論は古くから研究されている理論のひとつであるが，最近は電子計算機の発達にともなっていろいろな分野で応用されている．特に電気回路網理論の分野では，関数論的な立場に対して，グラフ理論的回路網理論といった分野が形成されつつある．これは回路網の結線構造に強く依存する性質を調べ，回路網の解析・合成に役だたせようとする動きである．

線形 RLC 回路網の入力インピーダンス・伝達インピーダンスなどの回路関数は，オームの法則とキルヒホッフの法則から連立方程式を導き，それを解いて得られる．節点解析またはカットセット解析と呼ばれる方法では，逆行列を求めるのに必要な行列の行列式の値が，回路網を描象化したグラフの木†††の枝にあるアドミタンスの積の総和になることが知られている[1]．この性質を用いると，連立方程式を解かずに回路網の接続の様子から直接回路関数を得ることが可能であり，位相幾何学的公式と呼ばれている．この公式の最大の特長は減算による相殺がないことである．

変成器，ジャイレータ，トランジスタなどを含む一般の回路網についても位相幾何学的公式が導かれており[2]，いずれの場合にも，グラフのすべての木を見出す方法が重要である．

電子計算機によって回路網の問題を解く場合に，回路関数を数値だけでなく，パラメータを含む記号式で得られれば，周波数領域の特性を調べる場合や，感度などの算出，あるいは理論的な研究にきわめてつごうが良い．この場合，行列式の展開や，連立方程式をSNOBOL のような記号処理言語を用いて強引に解くのも一つの方法であるが，複雑になるばかりでなく，計算に冗長が含まれ，適当ではない．釜江[3]，Lin と Alderson[4]などは，いくつかの素子あるいは周波数をパラメータとし，この部分に位相幾何学的な手法をあてはめ，他は単に数値的に解く方法を示したが，これらは記号として与えられる部分があまり多くないときに有用である．

回路関数を完全な記号式で得るためには，位相幾何学的公式を応用するのが適切である．そのためには，グラフのすべての木を見出す方法が重要であるが，グラフの木の総数はそのグラフの節点の数と比較すると指数的増加以上の割合で増すため，その手間は莫大である．したがって能率の良いアルゴリズムを示すことが必要であり，またアルゴリズムを実行するのに適し，しかもグラフという描象概念を能率良く扱い得る計算機言語でプログラムされなければならない．

本文ではこれらの目的のために，まず，基本となる無向グラフの木の集合について成り立つ性質を示し，この性質からすべての木を見出すアルゴリズムを導く．ここにおいて，グラフを並列な枝をひとまとめにした並列枝集合の和として表わし，この並列枝を操作してすべての木を見出すため，実行に要する手間は大幅に少なくなることを示す．グラフの位相幾何学的な性質を記憶し，枝の操作などを有効におこなうためには，二進木リスト構造を用いる．アルゴリズムは再帰

† A Method for Generating All Trees Suitable for Symbolic Network Analysis, by Chiaki HISHINUMA (Department of Electrical Engineering, Faculty of Engineering, Keio University)

†† 慶応義塾大学工学部電気工学科

††† 閉路を持たない連結グラフをいう．

的な情報の処理を含むので，LISP[5]のような再帰的関数処理とリスト処理のおこなえる計算機言語で簡潔に記述することができる．

2. 木集合定理

この章では，本文の基本となる定理と，これに必要な定義および性質を述べる．

考えるグラフはすべて無向グラフとし，$\mathbf{G}=(\mathbf{N}, \mathbf{B}, \partial)$ で表わす．ここで，$\mathbf{N}$ は節点 $\mathbf{n}_i$ の集合であり，$\mathbf{B}$ は枝 $\mathbf{b}_j$ の集合である．また ∂ は $\mathbf{B}$ から $\mathbf{N}$ の要素の順序を持たない対の集合 $(\mathbf{N}\,\&\,\mathbf{N})$ への関数であり，たとえば，$\mathbf{b}_j$ の両端の節点が $\mathbf{n}_p, \mathbf{n}_q$ であるならば，$\partial\mathbf{b}_j=\{\mathbf{n}_p, \mathbf{n}_q\}$ となる．

［定義1］ グラフの木は，直観的には閉路 (loop) を持たない連結グラフであって，$\mathbf{B}$ の要素からなる極大無閉路集合で定義される．さらに木をなす枝集合を要素とし，すべての木についての集合† を木集合と呼び，$\mathbf{T}$ で表わす．

［定義2］ グラフ $\mathbf{G}$ が与えられたとき，$\mathbf{G}$ の枝 $\mathbf{b}_j$ を取り除いて $\mathbf{b}_j$ 以外の枝はそのままにしておくことを枝 $\mathbf{b}_j$ を開放除去するという．また，$\mathbf{b}_j$ の両端の節点をまとめて新しい一つの節点とし，$\mathbf{b}_j$ を取り除いて他の枝はそのままにしておくことを枝 $\mathbf{b}_j$ を短絡除去するという．

［定義3］ 集合族 X と Y のすべての要素 x, y に対して，$w=x\cup y$ から成る集合 W を $X\otimes Y$ で表わす．すなわち，

$$W=X\otimes Y=\{w \mid w=x\cup y,\ x\in X,\ y\in Y\}.$$

連結グラフの木集合の性質を示すものとして，まずつぎの重要な定理を証明しておこう††．

［定理1］ 連結無向グラフ $\mathbf{G}$ の部分グラフ $\mathbf{G}_1$ と $\mathbf{G}_2$ が節点 $\mathbf{n}_p, \mathbf{n}_q$ のみを共有して並列接続をするとき (図1参照)，$\mathbf{G}$ の木集合 $\mathbf{T}$ はつぎのように表わされる．

$$\mathbf{T}=(\mathbf{T}_1\otimes\mathbf{T}_2')\cup(\mathbf{T}_1'\otimes\mathbf{T}_2). \qquad (1)$$

ここで，$\mathbf{T}_1, \mathbf{T}_2$ はそれぞれ $\mathbf{G}_1, \mathbf{G}_2$ の木集合であり，$\mathbf{T}_1', \mathbf{T}_2'$ は $\mathbf{n}_p$ と $\mathbf{n}_q$ を短絡してそれぞれ $\mathbf{G}_1, \mathbf{G}_2$ から得られるグラフの木集合である†††．

(証明) $\mathbf{G}$ の木をなす部分グラフには，$\mathbf{n}_p$ から $\mathbf{n}_q$ への道 $\mathbf{P}_{pq}$ が唯一つ存在し，それは $\mathbf{G}_1$ か $\mathbf{G}_2$ のどちらか一方に含まれる．したがって，$\mathbf{P}_{pq}$ を $\mathbf{G}_1(\mathbf{G}_2)$ に含む木の全体を $\mathbf{T}_a(\mathbf{T}_b)$ とすると，$\mathbf{T}=\mathbf{T}_a\cup\mathbf{T}_b$，かつ $\mathbf{T}_a\cap\mathbf{T}_b=\phi$ である．証明は，$\mathbf{T}_a=\mathbf{T}_1\otimes\mathbf{T}_2'$ および $\mathbf{T}_b=\mathbf{T}_1'\otimes\mathbf{T}_2$ となることを示す．

木 $\mathbf{t}_a\in\mathbf{T}_a$ の枝のうち $\mathbf{G}_1$ に含まれる部分を $\mathbf{t}_{a_1}$, $\mathbf{G}_2$ に含まれる部分を $\mathbf{t}_{a_2}(\mathbf{t}_a=\mathbf{t}_{a_1}\cup\mathbf{t}_{a_2},\ \mathbf{t}_{a_1}\cap\mathbf{t}_{a_2}=\phi)$ とすると，$\mathbf{t}_{a_1}$ は明らかに $\mathbf{G}_1$ の木であり，また，$\mathbf{t}_{a_2}$ は $\mathbf{G}_2$ 内で $\mathbf{n}_p$ から $\mathbf{n}_q$ への道を持たない $\mathbf{G}_2$ の枝の極大無閉路集合となるから，$\mathbf{G}_2'$ の木を構成する．したがって，すべての $\mathbf{t}_a$ について $\mathbf{t}_{a_1}\in T_1, \mathbf{t}_{a_2}\in T_2'$ となるから，$\mathbf{T}_a\subset\mathbf{T}_1\otimes\mathbf{T}_2'$ である．

逆に，すべての $\mathbf{t}_1\in\mathbf{T}_1$ と $\mathbf{t}_2'\in\mathbf{T}_2'$ に対し，$\mathbf{t}_1\cup\mathbf{t}_2'\in\mathbf{T}_a$ が示される．なぜならば，もし $\mathbf{T}_a$ の要素の部分木を成さない $\mathbf{t}_1$ または $\mathbf{t}_2'$ があるとき，これらを用いて $\mathbf{T}_a$ には属さないが，$\mathbf{n}_p$ から $\mathbf{n}_q$ への道が $\mathbf{G}_1$ に含まれる $\mathbf{G}$ の木を作ることができる．たとえば，$\mathbf{t}_1$ からは $\mathbf{t}_1\cup\mathbf{t}_{a_2}$，$\mathbf{t}_2'$ からは $\mathbf{t}_2'\cup\mathbf{t}_{a_1}$ が作られる．このことは，$\mathbf{T}_a$ が $\mathbf{P}_{pq}$ を $\mathbf{G}_1$ に含む木集合の全体であることに反する．したがって，$\mathbf{T}_a\supset\mathbf{T}_1\otimes\mathbf{T}_2'$ である．

したがって，$\mathbf{T}_a\subset\mathbf{T}_1\otimes\mathbf{T}_2'$，かつ $\mathbf{T}_a\supset\mathbf{T}_1\otimes\mathbf{T}_2'$ であるから，$\mathbf{T}_a=\mathbf{T}_1\otimes\mathbf{T}_2'$ となる．

$\mathbf{T}_b$ についても同様に示すことができ，定理の結果が得られる． (証明終)

定理1より容易につぎの関係が示される．

［系1］ $\mathbf{G}_2$ が一本の枝 $\mathbf{b}$ だけのグラフであるならば，

$$\mathbf{T}=\mathbf{T}_1\cup\{\{\mathbf{b}\}\cup\mathbf{t} \mid \mathbf{t}\in\mathbf{T}_1'\}. \qquad (2)$$

［系2］ $\mathbf{G}$ が図2のような両端の節点がそれぞれ一致したいくつかの枝の集合（並列枝集合）ならば，

$$\mathbf{T}=\{\{\mathbf{b}\} \mid \mathbf{b}\in\mathbf{B}\}. \qquad (3)$$

グラフの並列な枝を特徴的にとらえると，グラフの木集合に関して，より一般的な定理を導くことができる．そのためにまずつぎの定義をしておく．

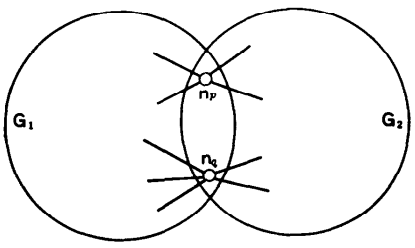


図1 並列なグラフ
Fig. 1 Graphs in parallel

† 正確には集合族である．
†† 文献(5)では代数的に表わし並列定理と呼んでいる．
††† $T_1'(T_2')$ は $G_1(G_2)$ の n_p n_q とをそれぞれ別の成分に持つ木集合に等しい．

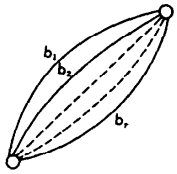
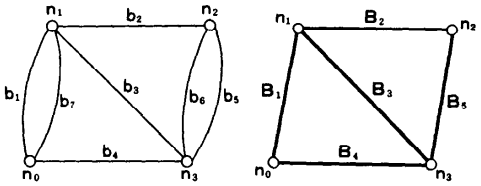


図 2　並列枝
Fig. 2　Parallel branches



(a)　(b)

図 3　(a)並列枝を含むグラフ
(b)新しく定義したグラフ
Fig. 3　(a) A graph with parallel branches
(b) Redefined graph

［定義 4］ 枝集合 **B** を並列枝集合の総和としてつぎのように定義する．

$$\mathbf{B}=\cup_{k=1}^{m}\mathbf{B}_k,$$

$$\mathbf{B}_k=\{\mathbf{b}\mid\partial\mathbf{b}=\{\mathbf{n}_{pk},\mathbf{n}_{qk}\},\ \mathbf{n}_{pk},\mathbf{n}_{qk}\in\mathbf{N}\},$$

$$\mathbf{B}_i\cap\mathbf{B}_j=\phi,\quad i\neq j. \tag{4}$$

すなわち，$\mathbf{B}_k$ は $\mathbf{n}_{pk}$ と $\mathbf{n}_{qk}$ を両端の節点とする枝の集合であり，**B** はこれらの m 個の集合の分割から成る．

このように枝集合を定義した無向グラフ $(\mathbf{N},\mathbf{B},\partial)$ の概念はつぎのように考えられる．たとえば，図3において(a)のグラフの中の並列な枝をひとつにまとめ，(b)のグラフとする．このとき，$\mathbf{B}_1$ には $\mathbf{b}_1$ と $\mathbf{b}_7$ の属性を保持させると，(a)のグラフの持つすべての性質が(b)のグラフで表わされる．

このようにグラフを並列枝集合の総和で表わすと，計算機内部にグラフを記憶する領域が小さくてすむばかりでなく，すべての木を見出す手間を大幅に少なくすることができる．この点については，**6.** で詳述する．

以上の定義のもとでつぎの定理を得る．

［定理 2］ グラフ $\mathbf{G}=(\mathbf{N},\mathbf{B},\partial)$ の任意の並列枝集合 $\mathbf{B}_k\subseteq\mathbf{B}$ に対して **G** の木集合 $\mathbf{T}(\mathbf{B})$ はつぎのいずれかを満足する．

(a) $\mathbf{B}-\mathbf{B}_k=\phi$ なるとき，

$$\mathbf{T}(\mathbf{B})=\{\{\mathbf{b}\}\mid\mathbf{b}\in\mathbf{B}\}. \tag{5a}$$

(b) $|\partial\mathbf{B}_k\cap\partial(\mathbf{B}-\mathbf{B}_k)|=2$ なるとき，

$$\mathbf{T}(\mathbf{B})=\mathbf{T}(\mathbf{B}-\mathbf{B}_k)\cup(\mathbf{T}(\mathbf{B}_k)\otimes\mathbf{T}(\mathbf{B}_i)), \tag{5b}$$

(c) $|\partial\mathbf{B}_k\cap\partial(\mathbf{B}-\mathbf{B}_k)|=1$ なるとき，

$$\mathbf{T}(\mathbf{B})=\mathbf{T}(\mathbf{B}_k)\otimes\mathbf{T}(\mathbf{B}-\mathbf{B}_k). \tag{5c}$$

(d) $\mathbf{B}-\mathbf{B}_k\neq\phi$，$\partial\mathbf{B}_k\cup\partial(\mathbf{B}-\mathbf{B}_k)=\phi$ なるとき，

$$\mathbf{T}(\mathbf{B})=\phi. \tag{5d}$$

ここで，記号 $|\ |$ は集合の要素数を示し，$\mathbf{B}_i$ は $\mathbf{B}_k$ を短絡除去したグラフで定義される枝集合である．また，$\mathbf{T}(\mathbf{B}_k)$ は $\mathbf{B}_k$ の枝からなる木集合を意味し，$\mathbf{T}(\mathbf{B}_k)=\{\{\mathbf{b}\}\mid\mathbf{b}\in\mathbf{B}_k\}$ である．

(証明) $\mathbf{B}_k$ を開放除去して **G** から得られる部分グラフを $\mathbf{G}'$ とすると，$\mathbf{B}_k$ と $\mathbf{G}'$ の連結関係は図4に示すような4つの型がある．(a) は **G** が互に並列な枝だけからなるグラフであり，(b) は $\mathbf{B}_k$ と $\mathbf{G}'$ が2節点を共有する場合，(c) は $\mathbf{B}_k$ と $\mathbf{G}'$ が1節点を共有する場合，(d) は $\mathbf{B}_k$ が他と共有部分を持たない場合である．これら4つの場合について，それぞれの木集合が定理の4つの式で示されることを証明する．

(a) $\mathbf{B}-\mathbf{B}_k=\phi$，すなわち **G** が一本の枝，あるいは互に並列な枝だけからなるグラフであるならば，それらの枝はそれ自身が一つの木であり，系2より式(5a)が成り立つ．

(b) $|\partial\mathbf{B}_k\cup\partial(\mathbf{B}-\mathbf{B}_k)|=2$，すなわち，$\mathbf{B}_k$ が $\mathbf{G}'$ に並列接続する場合，$\mathbf{B}_k$ を開放，短絡除去したグラフの木集合はそれぞれ，$\mathbf{T}(\mathbf{B}-\mathbf{B}_k)$，$\mathbf{T}(\mathbf{B}_i)$ で表わせるから．定理1より **G** の木集合は式(5b)で示される．

(c) $\mathbf{B}_k$ と $\mathbf{G}'$ が $\mathbf{B}_k$ の片端で接するときは，**G** のどの木も $\mathbf{B}_k$ の枝を必ず一本だけ含む．$\mathbf{G}'$ の木集合は $\mathbf{T}(\mathbf{B}-\mathbf{B}_k)$で示されるから，$\mathbf{T}(\mathbf{B})=\{\{\mathbf{b}\}\cup\mathbf{t}\mid\mathbf{b}\in\mathbf{B}_k,\ \mathbf{t}\in\mathbf{T}(\mathbf{B}-\mathbf{B}_k)\}$ となる．これより式(5c)が導かれる．

(d) $\mathbf{B}_k$ のほかに枝が存在し，$\mathbf{B}_k$ がそれらと接しないとき，グラフは非連結であり，木は存在しない．このときの木集合は空集合である．　(証明終)

3. アルゴリズム

定理2に示した木集合の性質は，容易にグラフのすべての木を見出す手続きとして表わすことができる．以下の説明では，グラフは定義4に従って並列枝集合の和で表わされるものとする．

木集合 $\mathbf{T}(\mathbf{B})$ を求めるアルゴリズム：

(i) 枝集合 **B** から一つの並列枝集合 $\mathbf{B}_k$ を取出す．

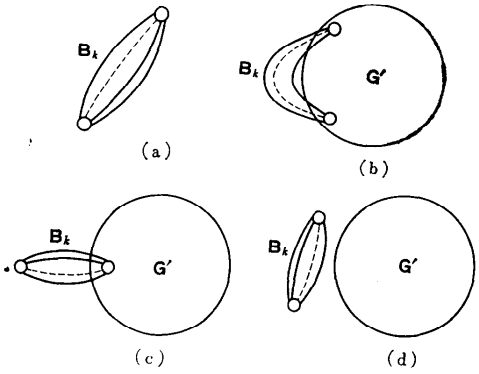


図 4 定理2の証明のための4つのグラフ
Fig. 4 4 illustrative graphs for Theorem 2

(ii) $\mathbf{B}_k$ 以外の並列枝の有無を調べる.

(iii) もし存在しないならば, $\mathbf{B}_k$ に含まれる各枝を木とする集合を値とする.

(iv) $\mathbf{B}_k$ 以外に枝が存在するとき, $\mathbf{B}_k$ の両端が $\mathbf{B}_k$ を除いたグラフ $\mathbf{G}'$ に含まれるかを調べる.

(v) 両節点とも $\mathbf{G}'$ に含まれるなら,

(v-1) まず, $\mathbf{B}$ から $\mathbf{B}_k$ を除いたグラフの木集合 $\mathbf{T}(\mathbf{B}-\mathbf{B}_k)$ を求める.

(v-2) つぎに, $\mathbf{B}_k$ を短絡し $\mathbf{B}_s$ を求める.

(v-3) $\mathbf{T}(\mathbf{B}_s)$ を求める.

(v-4) すべての $\mathbf{T}(\mathbf{B}_s)$ の要素と $\mathbf{B}_k$ の要素の和集合を要素とする集合 $\mathbf{T}(\mathbf{B}_k)\otimes\mathbf{T}(\mathbf{B}_s)$ を求める.

(v-5) $\mathbf{T}(\mathbf{B}-\mathbf{B}_k)\cup(\mathbf{T}(\mathbf{B}_k)\otimes\mathbf{T}(\mathbf{B}_s))$ を値とする.

(vi) $\mathbf{B}_k$ の一方の節点のみが $\mathbf{G}'$ に含まれるとき, $\mathbf{T}(\mathbf{B}-\mathbf{B}_k)$ を求め, この要素と $\mathbf{B}_k$ の要素の和集合を要素とし, これらのすべての組合せからなる集合 $\mathbf{T}(\mathbf{B}-\mathbf{B}_k)\otimes\mathbf{T}(\mathbf{B}_k)$ を値とする.

(vii) $\mathbf{B}_k$ の両端が $\mathbf{G}'$ に含まれないならば, 空集合を $\mathbf{T}(\mathbf{B})$ の値とする.

ここで注意すべきことは, 処理手順がそれ自身を含むいわゆる再帰的手続き (recursive procedure) になっていることである.

上記のアルゴリズムを実行するためには, つぎの5つの操作が基本となり, この手続きの実現方法が重要である.

操作1 並列枝集合 $\mathbf{B}_k$ を取出す.

操作2 $\mathbf{B}_k$ 以外の枝集合の有無を調べる.

操作3 $\mathbf{B}_k$ の両端の節点の一方あるいは両方が $\mathbf{G}'$ に含まれるかを調べる.

操作4 $\mathbf{B}_k$ の各枝を木とする木集合を求める.

操作5 $\mathbf{B}_k$ を短絡し, $\mathbf{B}_s$ を得る.

操作6 $\mathbf{T}(\mathbf{B}_k)\otimes\mathbf{T}(\mathbf{B}_s)$ をおこなう.

4. 条件式とリスト構造

再帰的手続きを定義するためには, McCarthy[7] の条件式 (conditional expression) の記法を用いると便利である. 条件式はつぎの形で書かれる.

$[p_1\rightarrow e_1;\ p_2\rightarrow e_2;\cdots;\ p_n\rightarrow e_n]$

ここで, p_i は真か偽の値をとる命題で, e_i は表現である. 条件式の値は p_1 が真ならば e_1 になり, もし p_1 が偽ならば, $[p_2\rightarrow e_2;\cdots;\ p_n\rightarrow e_n]$ となる. これは再帰的な定義になっている. さらに, $p_2, p_3\cdots$ と p_i が真になるところまで順々に調べ, 真である p_i が見つかったとき, e_i を値とする.

McCarthy は彼の作った計算機言語 LISP[5] にこの表現を用いるとともに, リスト構造† と呼ばれる記憶構造を導入した. 一般に, リスト構造は概念の包含関係や因果関係を主とする非数値データの記憶に適した構造で, グラフが持つ位相幾何学的な性質を記憶するのにきわめて適している.

グラフをリスト構造で記憶する方法はいくつか考えられるが, ここでは前述のアルゴリズムを実行するのに適したつぎのような構造を考える.

[定義5] まず互いに並列な枝 $\{\mathbf{b}_i, \mathbf{b}_j, \cdots, \mathbf{b}_k\}$ と, その両端の節点対 $\{\mathbf{n}_p, \mathbf{n}_q\}$ をつぎのようなリストで表わす.

$(N_p\ \ N_q\ \ B_i\ \ B_j\ \cdots\ B_k)$

さらに, グラフ全体は, この並列枝集合を表わすリストを要素とするリストで表わす. たとえば図3のグラフでは,

((N0 N1 B1 B7)(N1 N2 B2)(N1 N3 B3)(N0 N3 B4)(N2 N3 B5 B6))

となる.

グラフを表わすリスト 〈graph〉 の正しい定義は, バッカス記法を用いると, つぎのように示される.

〈graph〉::=(〈graph−list〉)

〈graph−list〉::=〈p−branch〉|〈p−branch〉〈graph−list〉

〈p−branch〉::=(〈node−list〉〈branch−list〉).

〈node−list〉::=〈node〉〈node〉.

† 正しくは二進木リスト構造 (binary list structure) であり, S式と呼ばれる表現形式で表わされる.

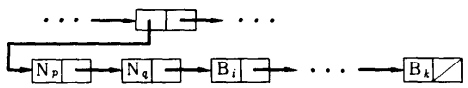


図 5 グラフを表わすリスト構造

Fig. 5 Representation of a graph by list structure

⟨branch−list⟩::=⟨branch⟩|
⟨branch⟩⟨branch−list⟩.

⟨node⟩::=節点の名前を示す原始記号.

⟨branch⟩::=枝の名前を示す原始記号.

この定義に基づくグラフを表わすリストは，計算機の内部で図5のように記憶される.

［定義6］ グラフの木集合を表わすリスト ⟨tree−set⟩ をつぎのように定義する.

⟨tree−set⟩::=(⟨tree−list⟩).

⟨tree−list⟩::=⟨tree−branch⟩|
⟨tree−branch⟩⟨tree−list⟩.

⟨tree−branch⟩::=(⟨branch−list⟩).

ここに，⟨branch−list⟩ は定義5によるが，各枝は木を構成する.

5. LISP 関数による定義

LISP はリスト処理と再帰的な関数処理の機能を備えた計算機言語であって，本文で述べたアルゴリズムを記述し，実行するにはきわめてつごう良い.

FORTRAN, ALGOL などの計算機言語は，「文」などで指示する仕事を順次処理するが，LISP では「関数」を定義し，それを評価する．LISP の関数はすべてリストを引数とし，値もリストである.

本章では，前章で定義したグラフを表わすリストを引数とし，これに対して木集合を表わすリストを値とするような LISP 関数が再帰的に定義できることを示す．そのためにまず **3.** の6つの操作を記述することから始める．以下の説明において，g は定義5に従って表わされるグラフ **G** を示すリストである.(LISP 関数の定義に関する詳しい説明はたとえば文献8）を参照.)

操作1　g の最初の項目 (car[g]) を $\mathbf{B}_k$ として選ぶ．こうすると，$\mathbf{B}-\mathbf{B}_k$ は cdr[g] で得られる.

操作2　$\mathbf{B}_k$ 以外に枝が存在すれば，$\mathbf{B}-\mathbf{B}_k \neq \phi$，存在しなければ，$\mathbf{B}-\mathbf{B}_k=\phi$ である．これらはそれぞれ，mul l[cdr[g]]=nil, null[cdr[g]]=t の場合である.

操作3　$\mathbf{B}_k$ の両端の節点は caar[g] と cadar[g] である．したがってこの場合の操作は，cdr[g] の中の caar[g] と cadar[g] の存在を調べればよい．グラフ y の中に節点 x が含まれるか否かを調べる述語関数† paratest はつぎのように定義できる.

```
paratest[x; y]=[null[y]→nil; eq[x;
    caar[y]]→t; eq[x; cadar[y]]→ t;
    t→paratest[x; cdr[y]]]
```

操作4　$\mathbf{B}_k$ の各枝からなる木集合を表わすリストは，

```
mapcar[cddar[g]; function[λ[[x]; list[x]]]]
```

で得られる††.

操作5　$\mathbf{B}_k$ を短絡する操作は，caar[g] と cadar[g] を一致させることである．関数 subst を用いると，

```
subst[caar[g]; cadar[g]; cdr[g]]
```

が $\mathbf{B}_k$ を短絡したグラフを示すはずであるが，短絡の結果，異なる枝が新たに並列になることがある．この部分については，正しく並列枝集合の和となるようにリストを変更しなければならない．この点を考慮して g を引数とし，$\mathbf{B}_i$ に相当するリストを値とする関数を定義すると，つぎのようになる.

```
gs[g]=gs 1[g: =subst[caar[g];
        cadar[g]; cdr[g]]]
gs 1[x]=[gs 2[x]→[cdr[x]→gs 1[cdr[x]];
        t→g]]
gs 2[y]=[null[cdr[y]]→t;
  or[and[eq[caar[x]; caadr[y]];
        eq[cadar[x]; cadadr[y]]];
     and[eq[caar[x]; cadadr[y]];
        eq[cadar[x]; caadr[y]]]]
  →prog 2[nconc[car[x]; cddadr[y]];
        gs 2[rplacd[y; cddr[y]]]];
  t→gs 2[cdr[y]]]
```

操作6　$\mathbf{T}(\mathbf{B}_k)\otimes\mathbf{T}(\mathbf{B}_i)$ は定義3より，$\{\mathbf{t}|\mathbf{t}=\{\mathbf{b}\}\cup\mathbf{t}',\ \mathbf{b}\in\mathbf{B}_k,\ \mathbf{t}'\in\mathbf{T}(\mathbf{B}_i)\}$ となる．この掛け算に似た操作は，つぎの関数によっておこなえる．ただし，始めに与えられる x と y の値はそれぞれ $\mathbf{B}_k$, $\mathbf{T}(\mathbf{B}_i)$ を表わすリストである.

```
multi[x; y]=mapcon[x; function[λ[[x];
    mapcar[y; function[λ[[y];
    cons[car[x]; y]]]]]]].
```

† 真または偽を値とする関数．条件式の命題になる.

†† ここでいう mapcar は次の定義による.

```
mapcar[x; f]=[null[x]→nil;
    t→cons[f[car[x]]; mapcar[cdr[x], f]]].
```

さて，以上の基本操作の実行に必要な副関数を用いると，グラフの木集合を求めるアルゴリズムはつぎのように LISP 関数で再帰的に記述できる．ここで，g はグラフを表わすリストであり，trees[g] の値はグラフのすべての木の集合を表わすリストである．

```
trees[g]=[null[cdr[g]]→mapcar[cddar[g];
  function[λ[[x]; list[x]]]];
paratest[caar[g]; cdr[g]]→[paratest[cadar[g];
  cdr[g]]→nconc[trees[cdr[g]];
  multi[cddar[g]; trees[gs[g]]]];
  t→multi[cddar[g]; trees[cdr[g]]]];
paratest[cadar[g]; cdr[g]]→multi[cddar[g];
  trees[cdr[g]]];
  t→nil]
```

6. 手間の評価

グラフの木の総数は節点の数に対して指数的増加以上の割合で大きくなるので，その見出す手間も莫大である．ここでは，本文に示した方法(並列枝方式)が，グラフの枝を一本ずつ操作して木集合を得る方法（単一枝方式）と比べて，はるかに手間が少ないことを示す．手間は，アルゴリズムが再帰的に実行される回数，実際には関数 trees が評価される回数とし，これらを比較する．

並列枝方式によって n 節点完全グラフ† のすべての木を見出すのに要する手間 f_n は，アルゴリズムの定義から，

$$f_n=1+f'^{\circ}_n+f'_n$$

と表わされる．ここに，f'°_n, f'_n は n 節点完全グラフの一本の枝をそれぞれ開放，短絡除去したグラフの木集合を見出す手間である．この場合，グラフは常に並列枝の和として表わす約束であるから，一本の枝を短絡除去して得られるグラフは $(n-1)$ 節点完全グラフである．したがって，

$$f'_n=f_{n-1}$$

であり，$n=2$ の場合は1回の手間で済むから，一般に，

$$f_n=\sum_{k=2}^{n}(1+f'^{\circ}_k)$$

と表わされる．f'°_k の値は，操作1における並列枝集合の選び方によって異なり，実際には，定義5によってリスト構造でグラフを表現する場合の枝の順序によって異なる．f'°_k が最小となるグラフの表現のしかたは，グラフの端の枝から順に互いに共有節点を持つように〈branch−list〉を構成することである．なぜならば，こうすることによって図4(d)の状態にならず，したがって，アルゴリズムにおける唯一の無駄な操作(vii)がおこなわれないからである．

単一枝方式では，その手間 g_n は同様に，

$$g_n=1+g'^{\circ}_n+g'_n$$

と表わせるが，n 節点完全グラフの一本の枝を短絡除去して得られるグラフは $(n-1)$ 節点完全グラフにいくつかの枝が重なったグラフになるので，

$$g'_n>g_{n-1}$$

となる．特に n が大きい場合，重なる枝も多くなるので，この差は大きくなり，一般に，

$$g_n\gg\sum_{k=2}^{n}(1+g'^{\circ}_k)$$

となる．

完全グラフの枝を一本取り除いたグラフに対し，アルゴリズムのつぎの段階における枝の短絡操作によって，単一枝方式では多重枝が生じ，並列枝方式では一つの並列枝に含められるのので，

$$f'^{\circ}_n>g'^{\circ}_n$$

であることがわかる．ただし，グラフを表わすのに〈branch−list〉の構成は全く同じであるとする．

以上のことより，単一枝方式と並列枝方式の手間を比較すると，

$$f_n\ll g_n$$

であることがわかる．

節点数 n の完全グラフに対する手間 f_n と g_n を表1に示す．単一枝方式の手間は木の総数の増加よりもさらに大きな割合で増えるが，並列枝方式では大幅に少ない．

完全グラフでない一般のグラフについても，枝の短絡操作によって完全グラフか，そうでなくても完全グラフに近くなるので，同様の評価をすることができる．

表1 完全グラフの木を求める手間の比較
Table 1 Comparison of the labors to find all trees in the complete graphs

節点数 n	木の総数 n^{n-2}	並列枝方式		単一枝方式		
		f_n	f'°_n	g_n	g'°_n	g'^{2}_n
3	3	**4**	2	**6**	2	3
4	16	**15**	10	**35**	18	16
5	125	**65**	49	**272**	165	106
6	1296	**337**	271	**2775**	1854	920
7	16807	**2072**	1734	**35532**	25385	10146
8	262144	**14794**	12724	——	——	——

† どの2つの節点間にも枝があるグラフをいう．

7. むすび

本文において連結無向グラフの木集合に関する定理を示し，これに基づいてすべての木を見出すアルゴリズムを導いた．この手続きは，グラフを二進木リスト構造で記憶し，再帰的な技法を用いて簡潔に表現することができる．また，グラフを並列な枝の集合として扱うと，木を見出すのに要する手間が大幅に少なくなることが示された．

本文で示された方法を応用すると，電気回路網の回路関数を完全な記号形で得ることができる．

謝辞　末筆ながら，本論文を書くにあたり多くの助言をいただいた本学管理工学科中西正和氏に深謝するとともに，日頃御指導賜わる末崎輝雄博士，森真作博士に感謝する次第である．

参考文献

1) S. Seshu and M. B. Reed: Linear graphs and electrical networks, Addison-Wesley (1961).
2) M. T. Jong and G. W. Zobrist: Topological Formulas for General Linear Networks,IEEE Trans. Circuit Theory, vol. CT-15, No. 3 p. 251, Sept. 1968.
3) 釜江：トポロジカルな手法をとりいれた線形回路網の解析法，電子通信学会，回路とシステム理論研資，CT 67-38, 昭和 43 年 1 月.
4) P. M. Lin and G. E. Alderson: Symbolic Network Functions by a Single Path-Finding Algorithm, Proc. Allerton Conference on Circuit and System Theory, 1969.
5) J. McCarthy, et al.: LISP 1. 5 Programmer's Mannual, The M. I. T. Press, Cambridge, Massachusetts, August 1962.
6) W. S. Percival: The Solution of Passive Electrical Networks by Means of Mathematical Trees, Proc. IEE, pt. 3, vol. 100, p. 143, May 1953.
7) J. McCarthy: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I, Comm. ACM, vol. 3, p. 184, April 1960.
8) 中西正和：KLISP 説明書改訂版，慶応義塾大学情報科学研究所，昭和 45 年 7 月.

（昭和 48 年 8 月 13 日 受付）
（昭和 48 年 9 月 7 日再受付）