オペレーティング・システム解析の一手法*

小松昭男** 本林　繁** 高橋延匡**

Abstract

This paper is concerned with a method of anlyzing and evaluating systems. The following two features are characteristics of this method; 1) An operating system is modeled in terms of a set of paths. (A path is a chain of nodes representing program modules.) 2) A technipue similar to GERT (Graphical Evaluation and Review Technique) is applied to this model in order to analyze and evaluate the performance of the operating system.

A program named OS-Analyzer has been developed with this idea. It consists of the following three phases; 1) Static analysis of the structure of the operating system by simply scanning it, 2) Dynamic analysis and modeling of the control transfers among the modules using the data obtained through the address-tracing, 3) Evalution and simulation of the program behavior by manipulating the model in conversational mode. Based on the results of these analyses, we can clarify various things about the operating sytem, e. g. structural and functional features, bottlenecks and efficiencies of reforming a module.

1. はじめに

最近の情報化社会において，電子計算機システムの果たす役割はきわめて重要である．今後，計算機システムの高度利用が進むにつれて，システムに対する要求は多種多様になる傾向にある．とりわけ，ソフトウェア・システムに対して，従来の機能の拡充だけの要素ではなく，その処理速度の増大，大きさ（ステップ数）の制限，コスト・パーフォーマンスの増大，作製期間の短縮など，改善しなければならない要素が多い．一方，OS (Operating System) の作製に関する工学はいまだ系統立てられているとはいえず，依然として“経験学”的な色彩が強い．このようななかで，OS をはじめとした，大型化・複雑化したソフトウェア・システムを整理・体系化する必要がある．本論においては，ソフトウェアの内部構造をネットワーク・フローで表現し，それに基づいたソフトウェアの評価・解析の手法について述べる．この手法の特徴的な考え方はネットワーク・フロー上のパス（ノードの系列）の集合として，ソフトウェア・システムをモデル化し，その評価・解析に OR (Operations Research) 手法を利用した点にある．これらはソフトウェア・システムの生産性向上の足がかりである．

なお本論においては，上述の手法を実際に応用したシステムである OS アナライザーを，H-5020TSS の OS の常駐部である MCP (Master Control Program) に適応して得られた結果およびその検討も述べてある．

2. OS アナライザーの考え方とその構成

2.1 概　　要

OS アナライザーは，OS の構造を2次元的に表現し，OS の実際の動作に基づいた定量的なデータをそれに重畳し，評価・解析するものである．その全体の概念図を**図 2.1** に示す．ここで，2次元的表現とは PERT (Program Evaluation and Review Technique) などで使われているネットワーク・フローに似たものであるが，プログラムの一般的性質上ループ (loop) を含んでいることに特徴がある．本論で扱うネットワーク・フローの例を**図 2.2** に示し，あわせて，ノード (node)，アーク (arc)，およびパス (path) の概念を示した．

このようなネットワーク・フローは一種のブロック図，またはステート・ダイヤグラム（状態遷移図）とみることができる．すなわち，OS のソース・プログ

* A Method of Analyzing Operationg System, by Akio Komatsu, Shigeru Motobayashi and Nobumasa Takahashi (Central Research Laboratory, Hitachi Ltd.)

** 日立製作所中央研究所

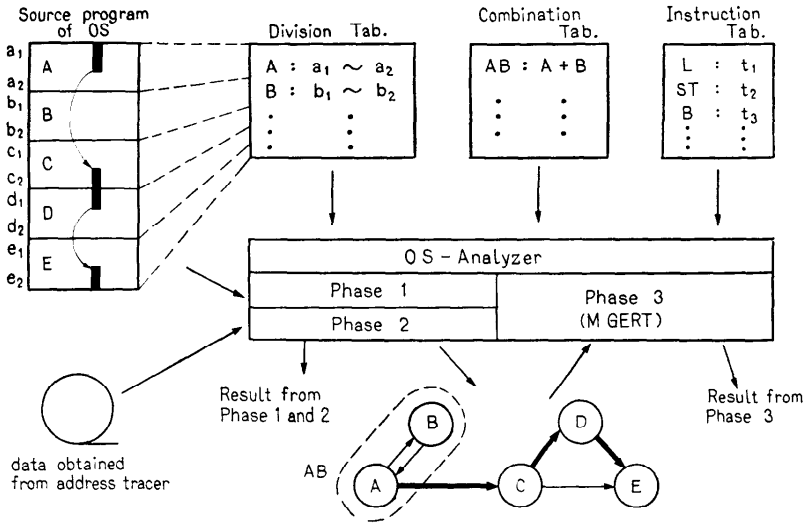
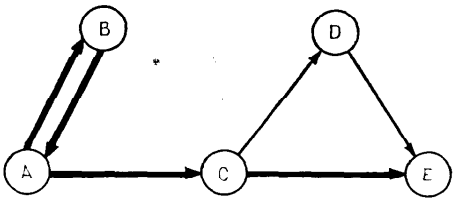


Fig. 1 Conceptual figure of OS-Analyzer

ラムをいくつかの機能単位(これをモジュールと呼ぶ)に分割し，各モジュールをネットワーク・フローのノードに対応させ，或るモジュールから他のモジュールへの制御の移動を，対応するノード間を結ぶアークとして表現することができる．

したがって，OS をモジュールに分割するための情報(具体的にはモジュールのアドレス範囲を示すもの)を参考にしながら，OS のプログラム（ラン・モジュール）を走査することにより静的 (static) なネットワーク・フローを描くことができる．ただし，この場合のアークは，モジュール間の制御の移動の可能性を示しているのであって，実際の OS の実行に際して対応する制御の移動が生じない場合もある．このような意味で，このネットワーク・フローを静的ネットワーク・フロー (Static Network Flow: 以下 SNF と略す）と呼ぶ．

次に，OS の実際の動作の軌跡であるアドレス・トレース，またはハードウェア・モニターなどの活用によって，ノードの系列の集合を得ることができる．ここで，ノードの系列とは，OS がある処理をするために生じたモジュール間の制御の移動の歴史であり，これをパス (path) と呼ぶ．たとえば，OS における一つの割込み処理の過程は一つのパスに対応する．さらに，種々の SVC (Supervisor Call) 命令——OS に対する処理要求——とその処理過程（すなわちパス）とは1対1に対応するものである．このようなパスの



Circles (nodes) represent each program modules, and arrows (arcs) represent intermodule transmission of control. A path is a chain of nodes. (e. g. ABACE)

Fig. 2.2 Networkflow representation of a program

集合を整理した統計量に基づいて，動的ネットワークフロー (Dynamic Network Flow: 以下 DNF と略す）を得ることができる．

DNF は一種の確率的ネットワーク・フローとみることができ，各モジュールの平均実行時間とモジュール間の遷移確率とを含んでいる．このような DNF に対して，PERT や GERT (Graphical Evaluation and Review Technique) などのネットワーク理論をはじめとする OR (Operations Research) を活用した OS の評価・解析が可能になる．さらには，OS のモデルとしての DNF に基づいた簡単なシミュレーションが可能である．

以上が OS アナライザーにおける考え方である．今回試作開発した OS アナライザーへの入力は，OS の実体である実行可能なプログラム（ラン・モジュー

ル)，OS の実際の動作の軌跡であるアドレス・トレース，および補助入力としての3つのテーブルである．OS アナライザーは3つの段階 (Phase) にわかれていて，各々の段階について2.3節以降にて詳しく述べる．

2.2 参照テーブル

OS アナライザーが補助入力として参照するテーブルは以下の三つである．

(1) 分割情報テーブル

OS 全体をいくつかの機能単位に分割するための情報である．具体的には，モジュールごとにその先頭アドレスと最終アドレスとを定義したものである．したがってこのテーブルは，OS の機能分割に対しての一つの試みであると考えられる．

(2) 結合定義テーブル

一般的な OS の機能分割はきわめて heuristic であり，一意的には決定できない．このため，種々の OS に対しての機能分割を試み，それらの積み重ねによって汎用性のある機能分割を考える必要があり，分割情報テーブルのみでは柔軟性に欠ける．この点を補うものとして結合定義テーブルを設けた．これは，分割されたいくつかのモジュールをまとめて一つのモジュールとして再定義するものであり，次のような利点がある．第一に，分割情報テーブル作製においては，対象とする OS に関しての知識（ソース・プログラムの解読能力など）が必要であるが，結合定義テーブルの作製においては，OS の機能の概要を理解していればよい．したがって，OS 作製者などの助言に基づいて分割情報テーブルを作製し，このテーブルは固定のまま，以後の解析ごとに結合定義テーブルを変化させ，各種のレベルでの OS の機能分割に関しての資料を得ることができる．第2に，分割情報テーブルでは離れたアドレス範囲をもつ部分を一つのモジュールに定義することはできない，という欠点を補うことができる．

(3) 命令テーブル

対象とするプログラムに使用されている命令の種類とその命令実行時間を定義するものである．命令の使用頻度や実行頻度，および OS ミックスやモジュールの走行時間などの測定に用いる．

2.3 SNF の作製（フェーズ 1）

2.2 節で述べた3つのテーブルを参照しながら，OS のラン・モジュールを走査して，SNF (静的ネットワーク・フロー) に関する情報を得るまでの段階がフェーズ1である．フェーズ1における解析出力として次のものがある．

(1) SNF を描くための情報

OS の各モジュール間の制御の移動の可能性を示す情報であり，一般の流れ図に準じた情報である．この情報に基づいて SNF を描くことができる．

(2) 命令の使用頻度

OS を作製するために使用された命令の頻度を，命令テーブルに基づいて計数したものである．(したがって，実行時における命令実行頻度とは異なる．) この値によって，OS のプログラムにおいて，どのような命令がどの程度使われたかを知ることができる．

2.4 DNF の作製（フェーズ 2）

フェーズ1で得られる SNF に，実際の OS の動作 (OS のアドレス・トレース) に基づいて得られるデータを重畳して，DNF (動的ネットワーク・フロー) に関する情報を得るのがフェーズ2である．もちろん，フェーズ1を使用せずに，最初からフェーズ2を適用して，OS の動的 (dynamic) な解析の結果を得ることができる．フェーズ2においては以下の解析項目を出力する．

(1) 相異なる種類のパスとその出現回数
(2) 各パスの実行ステップ数と実行時間．
(3) パスの出現順序．
(4) 各モジュールの出現回数．
(5) 各モジュールの平均実行ステップ数と平均実行時間．
(6) モジュール間遷移頻度（または遷移確率）．
(7) 命令実行頻度およびそれに基づく OS ミックス．

2.1 節において述べたように，一つのパスは OS の一つの処理内容に対応していること，および各モジュールは OS の各機能単位に対応させていることなどを考え合せれば，上記の(1)から(6)までの解析項目は次のような意味をもつことが容易に理解できる．

(1)′ OS がどのような種類の処理をどの程度の頻度で処理しているかを，記号別（すなわち，モジュール名の系列であるパス）の区別とその出現回数の計数とを，機械的に行なうことによって把握することができる．さらに，OS の一つの処理の実行において，どのような機能単位（モジュール）がどのような順番で実行されたかを理解することができる．

(2)′ OS が各種の処理を行なうための時間を知り得る．

(3)′ パスの出現順序はOSに対する処理要求の出現順であって，OSへの負荷であり，OS利用者の行動 (behavior) の一つのモデルである．

(4)′ OSの各機能単位の利用頻度を知り得る．

(5)′ 各機能単位を実行するに要する平均的な時間とステップ数を意味している．

(6)′ 各機能単位間の制御のやり取りを定量的に示したものであり，遷移頻度（または遷移確率）に基づいて，OSを一種の確率的ネットワーク・フローとして表現することができる．

2.5 解析と評価のプログラム（フェーズ3）

DNFはOSの実際の動作をも反映させた簡潔な表現であって，OSのレントゲン写真であると考えられる．これに基づいた種々の診断（解析）を行なうのがフェーズ3である．すなわち，DNFに対してOR手法を適用した簡単なシミュレーションを行ない，OSを評価・解析するものである．

2.5.1 フェーズ3とGERT手法

フェーズ3では，GERT[2),3)]に似た手法によって，フェーズ2で出力されたデータを種々の側面から解析するものである．まず，GERTとの違いを含めてフェーズ3の考え方をあきらかにするために，GERTについて多少ふれておく．

OR手法のなかで良く知られたものにPERTとかCPM (Critical Path Method) などがある．このようなネットワーク理論は，種々の物理現象（交通・通信・輸送など）に対して有効な数学的モデルを提供している．しかし，その適用分野を制限するものとして次の二つの前提条件が考えられる．

(1) ネットワーク・フローはループを含まないこと．

(2) ネットワーク・フローの各ノードはAND入力，AND出力であること．

このような面を解決するものとして，GERTにおいてはネットワーク・フローの性質を確率的なフローに拡張している．GERTには次のような特徴がある．

(1) ネットワーク・フローのなかにループを含んでいてもよい．

(2) 各種の論理的なノード (logical node) を含む．

(3) 確率変数(任意の分布を許す)を導入できる．

(4) sensitive analysis（確率変数の変化が出力結果に与える影響の解析）が可能である．

すなわちGERTとは，与えられた確率的ネットワーク・フローに対して，等価変換の規則に従ってネットワーク・フローをreduceしていく技法である．結局，GERTにおける入力は各ノードの平均実行時間の分布と次のノードへの遷移確率である．しかし，OSアナライザーのフェーズ3においては，GERTを計算機システムの評価のために，その入力として次のものを選ぶように改良を行なった．

(1) 各ノードにおける平均実行時間（および，平均実行ステップ数）．

(2) 各種のパスとその出現回数（重みづけられたパスの集合）．

ここで，各ノード間の遷移確率のかわりに重みづけられたパスの集合を選んだのは次のような理由による．すなわち，確率的ネットワーク・フローにおいては，ノードの系列という考え方には不向きであり，ノードの前後関係およびノードの2字組，3字組，…に対しては確率的にしか論じられない面がある．しかしフェーズ3においては，OSのプログラムをいくつかの機能に分けてノードに対応させ，それらのマクロ的な機能としてのn字組を考えることが容易である．さらには，OSの処理内容と1対1に対応したパスの概念を有効に活用することができる．このように，フェーズ3は確率的ネットワーク・フローの表現方法においてGERTとは異なっている．以後これをMGERT (Modified GERT) と呼ぶことにする．

2.5.2 フェーズ3の概要

フェーズ3の処理方式としては，試行錯誤で解析やシミュレーションができる会話型処理が適していると判断し，MGERTはH-5020 TSSのPL1Wで開発した．なお，MGERTはフェーズ1やフェーズ2とは独立に使用できるようになっており，入力として2.5.1で述べたような表現による確率的ネットワーク・フローを与えることによって，種々の物理現象を解析することもできる．

フェーズ3によって解析できる項目は，MGERTで準備したコマンド（H-5020 TSSからみればサブ・コマンド）の処理内容に依存する．現在は次に示すような10個のコマンドが使用できる．

GRAPH: 個々のモジュールのすぐ次の後続モジュール名とその遷移確率とを求める．

SEARH: 与えられた改善率（%）に基づいて，改善効率のよいモジュール名5個と各々の改善効率を出力する．

COM: いくつかのモジュールをまとめて1つの

モジュールに再定義する.

URDAT: モジュールやパスの統計量を変更する.

TOTAL: 内部のテーブルを整形し，全体の統計量を出力する.

PATH: パスに関する統計量を出力する.

MODUL: モジュールに関する統計量を出力する.

KUMI: 与えられた n 字組の出現すパスの番号とその出現回数を出力する.

DESND: 与えられたモジュールの後続モジュール名をすべて出力する.

END: MGERT の処理を終了させる.

これらのコマンドを使ってOSに対する負荷が変った時とか，OS の部分的な修正を行なった場合に，OSの実行時間の期待値がどのように変化するかをシミュレートすることができる．さらに，改善効率のよいモジュールの検出も可能である．ここで改善効率とは，あるモジュールの平均実行時間を短縮させた場合のOSにおける全体の実行時間の期待値の減少度をいう．すなわち，次のように数式的に定義される．

M_{ik} を i 番目のパスの上の k 番目のモジュールの平均実行時間とすると，i 番目のパスの実行時間 P_i は次式で求まる．

$$P_i=\sum^{k} M_{ik}.$$

W_i を i 番目のパスの出現頻度（重み）とすると，OS 全体の実行時間の期待値 E は次式で求まる．

$$E=\frac{\sum^{i} P_i W_i}{\sum^{i} W_i}.$$

ここで，M_{ik} が M_{ik}' に変化した時の E の値が E_1 から E_2 に変化したとすると，この時の改善効率 I は次のように定義される．

$$I=\frac{E_1-E_2}{E_1}\times 100.$$

さらに，モジュールの改善率は，$(M_{ik}-M_{ik}')\times 100/M_{ik}$ で定義する．

3. 結果とその検討

OS アナライザーは，これまで述べたOSの解析手法を実際に応用して試作開発を行なったシステムである．本章においては，OS アナライザーを実際のOSに適応して得られた結果とその検討について述べる．対象としたOSは，H-5020 TSS[4]の常駐部であるMCP (Master Control Program) と呼ばれているプログムである．なお，OS アナライザーへの入力となるアドレス・トレース情報は，既に開発されていたPATTERN[6]と呼ばれるトレーサーの出力を利用した．

3.1 OS のネットワーク・フロー表現

PL/I のようなシステム・プログラム記述用言語によるOSの表現は，1次元的であってその構造が理解し難い面がある．これに対して，ネットワーク・フロー表現では構造が視覚的に理解できる利点がある．一方，言語による記述はOSの各部分の機能を正確に記述できることに対して，ネットワーク・フローによる表現では，各ノードに対応したモジュールの機能を一般的に理解できる内容にしなくてはならない．すなわち，一つのノードに一つのルーチン（システム・マクロなど）を対応させて，そのルーチンの名前をノードの名前として使い，この名前によってそのノードに対応する機能を理解することである．このような方法によって，OS のネットワーク・フロー表現によって，構造ばかりでなく，OS の各部分の機能の関係をも視覚的に理解でき，OS の簡潔なドキュメンテイションとしても利用できる．

約 8K 語（32 bits/語）の大きさをもつ MCP の場合には，分割情報テーブルによって約250個のモジュールに分割した．平均すると，一つのモジュールの大きさは約 32 ステップということになる．（モジュールをこのように小さく選んだのは，2.2 節で述べた結合定義テーブルを有効に活用するためである.）これらのうち，アドレス・トレースに出現したモジュールは137個（54%）であった．出現しなかったモジュールは，（1）トレースされなかった特殊な SVC (Supervisor Call) の処理に関するモジュール，（2）特殊な割込処理やエラー処理のためのモジュールなどである．これらの出現したモジュール間の関係をネットワーク・フローで表現したとすると，アークの数は 343 個でかなり複雑なフローになってしまう．（このアークの数は実際にフローを描いた結果として求めたものではなく，フェーズ3の出力としてフローを描いた場合のアークの数を出力したものによる.）結合定義テーブルなどの活用により，モジュールの数を 56 個にまとめた場合のアークの数は158個であった．

また，フェーズ3より**図 3.1** に示したようなモジュール間の遷移確率が求められる．（図 3.1 に示したのは，ENTR と名づけたモジュールに関しての出力である．小数点以下は切り捨てたので合計は100%にはならない.）これらによって MCP のモジュール間の

```
*** COMMAD ***……GRAPH
ALL OR XXXX (MNAME) / ……ENTR
BACKWARDREF. TAB.
ENTR 1696   TRCS   1%    MFAN   30%    BSV    1%
             149   1%     141    3%   ACNT    0%
            MTST   3%    CHSD   31%   PGFR    8%
             147   2%     PRT    0%    125    0%
            SOFT   0%     DPR    0%   SDET    2%
             124  11%
```

Fig. 3.1 Transmisson ratio of control from a program module 'ENTR'

動的ネットワーク・フローに関する定量的なデータを詳細に把握できる．なお，これらのデータから図的にネットワーク・フローを描くことができるが，依然として複雑であり，各モジュールの配置などを考えてフローに描くにはかなりの労力を要するであろう．なお，XYプロッターなどを使って簡潔なフローを自動的に描かせるためのアルゴリズムは複雑であり，そのためのプログラムの開発は今回は省略した．

3.2 OS の処理内容

フェーズ2で出力される各種のパスに関するデータの検討やフェーズ3の活用によって，OS の処理内容を解析することができる．

MCP において，そのアドレス・トレースに含まれていたパスの種類は約 90 種類であった．このなかには，本質的には同じ処理内容であるが，その処理内容の一部が若干異なるために異なったパスとして計数されたものが含まれている．たとえば，同じミッシング・フォールトのための処理でありながら，ミッシング・フォールトの原因がページであったか，ページ・テーブルであったかによって，パスの中の2～3のモジュールが違った場合である．これらの似たような処理内容を結合定義テーブルなどを利用してまとめると，パスの種類は 56 種になった．これらのパスの分類の結果は**図 3.1** より知ることができる．すなわち，ENTR と名づけられたモジュールは MCP の入口であって，ここで各種のデータが整形された後対応する処理モジュールに制御が移っている．したがって，ENTR モジュールから後続モジュールへの遷移頻度によって MCP の処理内容を分類できる．図3.1において，CHSD（チャネル・スケジューラ）モジュールへの遷移頻度が一番多く，31%であった．これはこのデータのもととなったアドレス・トレースにおいて，システムの立上りの情況も含まれていたため，ディスクやドラムからのプログラムのロードが頻発したためと考えられる．その次に高い遷移頻度であったのは，MFAN（マッピング・フォールト・アナライズ）モジュールで，30% であった．

このような資料は，H-5020 TSS のようなオン・デマンド・ページング方式において常に問題とされているミッシング・フォールトに関して考察する場合の貴重な資料となる．すなわち，ミッシング・フォールトの処理に対応した各パス，およびそのパス上の各モジュールの統計量を他のシステム・データとともに分析することである．また，ミッシング・フォールトのための処理のうち，長いもので2,000 ステップスも要しているものがある．この数値が果して妥当なものかどうかは，他の同じ方式の OS における結果と比較・検討しなくてはならない．これらの検討事項は今後の問題として取組んでいきたい．

3.3 モジュールの改善効率

OS を作製することと共に OS のネックの抽出およびその改善を効率よく実行することも大切なことである．したがって，プログラムのどの部分をどの程度改善すれば，全体としてどの程度改善されるかを定量的に把握することが必要である．これに対して，フェーズ3の簡単なシミュレーションによって，各モジュールの改善効率を定量的に調べることができるようになった．改善効率については2.5節にて述べた．

フェーズ3を適用して，MCP の 138 個のモジュールの中から改善効率のよいモジュールを5個検出してみた．このなかの上位2つのモジュールに関してグラフにまとめたのが**図 3.2** である．図 3.2 からわかる

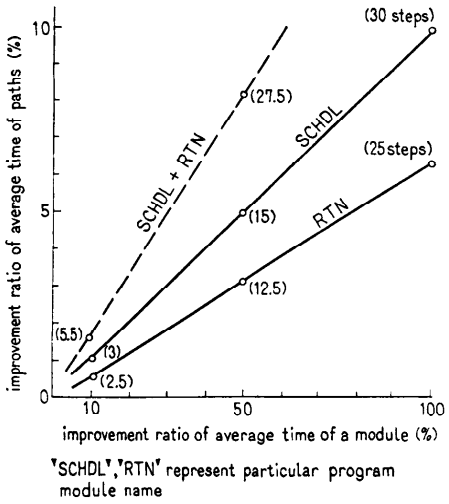


Fig. 3.2 Improvement ratio of average time of paths by improving average time of a module

ことは，一番改善効率のよいモジュールの場合，改善効率はモジュールの改善率の 1/10 程度である．SCHDL という名前のモジュールは，プロセス・スケジューリングを行なっている部分（したがって通過頻度が最も高いモジュールのうちの一つ）である．このモジュールの大きさは 103 ステップであるが，その平均実行ステップ数（このモジュールに入ってから出るまでに実行されるステップ数の平均値）は約 30 ステップである．平均実行時間と平均実行ステップ数とが比例するとして，SCHDL モジュールの平均実行ステップ数を 10%（3～4 ステップ）減少させることによって，MCP 全体の実行時間の期待値が 1% 減少することになる．2番目の RTN モジュールは MCP から制御が抜け出すところであり，レジスター類の回復が主な仕事である．したがって，このモジュールをすべてハード化させたとすれば，モジュールの改善率は 100% となり，6% 以上の改善効率が期待される．なお，これらのデータは，各モジュールを単独に改善した場合であって，二つのモジュールを同時に改善した場合は，図 3.2 の破線のように改善効率はさらに向上する．

上記のデータに関して二つの見方がある．第一に悲観的な見方である．すなわち，3～4 ステップの減少は平均実行ステップ数に対してであって，物理的な大きさ（今の場合には 103 ステップ）を 3～4 ステップ減らすことではなく，実際に改善すべきステップ数はもっと多いかも知れないこと．さらに，これらは統計量であって，それがそのまま現実のものとはなり得ないかも知れない，という見方である．これに対して，該当するモジュールの処理内容を考えれば，3～4 ステップの短縮は簡単であるという見方がある．すなわち，そのモジュールのなかによく使われるループなどがあれば，ループ内の 1 ステップでも短縮させれば，結果的に平均実行ステップ数を 3～4 ステップ短縮させたことになる．また，他のモジュールも同時に改善すれば，MCP 全体の改善率はさらによくなることが期待される．（得られたデータに基づいて計算すると，5 個すべてを 10% ずつ改善すると全体として 2.5% 改善されることになる．）

いずれにしろ，プログラムの改善は，改善されるべきプログラムの作り方や内容に依存するので一般的には論じられない．しかし，このように定量的データに基づいてプログラムの改善を行なうことは従来では不可能であった．これに対し，OS アナライザーを客観的な助言者としてプログラムの効率のよい改善に役立つものと考える．

以上のものの他に，OS アナライザーによって付随的に得られるものとして，各命令の実行頻度に基づく OS ミックスや，OS とユーザーとのプログラムの走行ステップ数の比較などの結果がある．これらについては紙面の都合で省略する．

4. おわりに

ソフトウェアー・システムの評価・解析の一つの手法として，プログラムをネットワーク・フローにおけるパスの集合としてモデル化し，その評価や解析に，OR 的手法を活用することを試みた．今回開発した OS アナライザーを実際の OS（H-5020 TSS の OS の常駐部である MCP）に適応することにより，MCP の動作や処理内容に関してのデータを得ることができた．さらに，本論で述べた評価・解析手法の有効性を確信することができた．

なお，OS アナライザーにおいては，このような解析のもととしてアドレス・トレース情報を活用している．当初，システムの動作のほんの一部の拡大図でしかないアドレス・トレース情報によって，システム全体を評価することに対する不安はあった．しかしながら，上述の MCP に適応して得られた結果より，次のような確信を得た．すなわち，各パスに対する重みを適当に加減することによってシステム全体の動作を忠実に反映させることができることである．事実，MCP に関して得られた各パスの重みを加工することなくそのまま解析した結果は，OS アナライザー以外によって得られたデータとよく一致していた．しかしながら，アドレス・トレース情報を得るための時間的および空間的（磁気テープなどの容量）問題などがある．このため，今後はハードウェア・モニタなどの活用を考える必要があると思う．

謝辞 本研究に関して有益な討論や助言をいただいた益田隆司氏，吉沢康文氏，広沢敏夫氏，および日立中研 9 部 105 ユニットの皆様に深く感謝する．

また，日ごろ熱心な御指導をいただく日立中研嶋田正三主管研究員と田上嵩第 9 部部長に対し，つつしんで感謝の意を表する．

参 考 文 献

1) 三根久：“オペレーションズ・リサーチ，” 朝倉書店，東京 (1966).

2) S. E. Elmaghraby: "An Algebra for the Analysis of Generalized Activity Networks," Management Science, Vol. 10, No. 3 (1964).
3) B. Beizer: "Analytical Techniques for the Statistical Evaluation of Program Running Time," FJCC, pp. 519～524 (1970).
4) 高橋，益田: "HITAC 5020-TSS の解析と評価，" 情報処理学会 OS シンポジウム報告集, pp. 6～36 (1970).
5) L. A. Belady: "A Study of Replacement Algorithms for a Virtual Storage Computer," IBM Sytem Jour., Vol. 5, pp. 78～101 (1966).
6) 益田，高橋，吉沢: "ページング・マシンにおけるスワッピング・アルゴリズムの比較とプログラムの動作解析，" 情報処理, Vol. 13, No. 2, pp. 81～88 (1971).

(昭和47年4月5日受付)
(昭和47年6月13日再受付)