



Title	プログラミング技法に関する研究ならびにデバイス シェアリングシステムの作製
Author(s)	葛山, 善基
Citation	大阪大学, 1976, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/2438
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

プログラミング技法に関する研究
ならびに

デバイスシェアリングシステムの作製

葛山善基

工27

39

3624

内 容 梗 概

本論文は筆者が大阪大学大学院基礎工学研究科（物理系専攻）の学生として、當研究室において行なった情報処理に関する研究のうち、プログラミング技法に関する研究ならびにデバイスシェアリングシステムの作製を四章にまとめたものである。

緒論および各章の第1節では研究の現状，その工学上の意義，本研究の新しい諸成果について概説している。各章の最後の節および全体の結論では，本研究で得られたおとな結果と今後に残された問題について述べている。

正確かつ効率のよいプログラムを能率よく作製することとは重要なことである。

第1章では，ミニコンピュータPDP-11/20のデュアルシステムの入出力装置を両システムから効率よく使用するデバイスシェアリングシステムについて述べている。

第2章では，有向グラフで表現されたヤーノフのプログラム形が与えられたとき，それと強等価でかつ節点数の総和が最小であるという意味で最簡のヤーノフのプログラム形を求める手続きについて述べている。

第3章では，ヤーノフのプログラム形が与えられたとき，多入力を許したサブルーチンを用いて，動作等価な部分をまとめることにより，スタック節点，リターン節点以外の節点数の総和が最小であるプログラム形を求める能率のよい手続きを述べている。

第4章では，サブルーチンなどのモジュール間のコール関係と影響関係が与えられたとき，できるだけ多くのモジュール構成を保ち，影響範囲を子孫，親，兄弟のモジュール内に局限されるように，もとのモジュール群を合併する能率のよい手続きを述べている。

プログラミング技法に関する研究ならびに デハイスシェアリングシステムの作製

目 次

緒 論	-----	1
第 1 章	PDP-11/20 デュアルシステムにおける デハイスシェアリングシステム	9
1.1	序 言	9
1.2	既存システムの概要	12
1.2.1	ハードウェア	12
1.2.2	DOS モニタ	15
1.3	システム作製の基本方針	17
1.4	データ転送	18
1.5	デハイスのロック・アンロック	20
1.5.1	ロック方式	20
1.5.2	ファイル構造を持たないデハイス	22
1.5.3	ファイル構造を持つデハイス	24
1.6	デハイス管理ルーチン (DEMON)	27
1.6.1	DEMON の機能	27
1.7	結 言	35
第 2 章	ヤーフのプログラム形の簡単化	37
2.1	序 言	37
2.2	定義と問題	39
2.3	最簡形 S_{min} を求める手順	42
2.4	定理の証明	44
2.5	結 言	47

第 3 章	サフル - 4 ン化によるプログラム形の単純化	49
3.1	序言	49
3.2	定義と問題	51
3.3	サフル - 4 ン可能グラフ	58
3.4	サフル - 4 ン化の条件	59
3.5	R 内の最簡形の構成アルゴリズム	61
3.5.1	S から $\bar{S}$ への変換	61
3.5.2	$\bar{S}$ の各節点 π のサフエンドを求める	61
3.5.3	$\bar{S}$ の節点の分割 $\pi(\bar{S})$ を求める	62
3.5.4	$\bar{S}$ の定義	64
3.6	$\bar{S}$ の等価性とコスト最小性	66
3.7	結言	68
3.8	付録	69
第 4 章	プログラムモジュール群の合併	73
4.1	序言	73
4.2	定義と問題	76
4.3	G が有向木である場合	78
4.3.1	カット枝集合を求める問題への変換	78
4.3.2	最大カット枝集合を求める手順の説明	82
4.3.3	最大カット枝集合を求めるアルゴリズム	89
4.4	結言	92
結 論		93
謝 辞		96
文 献		97

緒 論

オペレーティングシステムをはじめ各種のプログラムの規模が増大するにつれ、正確かつ効率のよいプログラムをいかに能率よく作製するかが重要な問題となっている。

本論文はプログラミング技法の研究ならびにデバイスシェアリングシステムの作製について、次に示す4つのことをまとめた。〔Ⅰ〕PDP-11/20 デュアルシステムに対するオペレーティングシステムで、デバイスの占有時間が不当に長くならないう意味でハードウェアの使用効率が良いデバイスシェアリングシステムに関する研究ならびに作製。〔Ⅱ〕有向グラフで表現されたヤーフのプログラム形が与えられたとき、それと同等価でかつ節点数の総和が最小であるという意味で最簡のヤーフのプログラム形を求めることに関する研究。〔Ⅲ〕ヤーフのプログラム形が与えられたとき、多入力サフルーチンを用いて、動作等価な部分をまとめることにより、スタック節点、リターン節点以外の節点数の総和が最小であるプログラム形を求めることに関する研究。〔Ⅳ〕サフルーチンなどのモジュール間のコール関係と影響関係が与えられたとき、できるだけ多くのモジュール構成を保ち、影響が与えられる範囲がその影響を与えるモジュールの子孫、親、兄弟のモジュール内に局限されるようにもとのモジュール群を合併することに関する研究。

第1章には、高速インタフェースプロセッサを介して接続されたミニコンピュータPDP-11/20 2セットに対し、筆者が中心となって開発したソフトウェアシステム（デバイスシェアリングシステムと呼ぶ）について述べている。本システムは、文献(1)、(2)、(3)として公表した。

本システムは、既存のディスクオペレーティングシステムDOSを改造して、コンソール・タイプライタとテレタイプライタを除く全ての入出力装置（外部記憶装置を含む）が効率良く、しかもあたかも自システムに接続されているかのように使用することができるようにしたものである。

DOSにはマルチプログラムの機能がなかったので、デバイス・シェアリング・システムを作製するにあたり、ファイルに対する相互排除が大きな問題となった。この章では、①入出力装置と主記憶とのデータ転送の方式、②ファイル構造を持たない入出力装置の相互排除の方式、③ファイル構造を持つ入出力装置の相互排除の方式、④相互排除要求を処理するデバイス管理ルーチンについて述べている。

このデバイスシェアリングシステムを作製するにあたりDOSを分析したが、プログラミング技法として注目されるのはプログラムの長さをできるだけ短かくするための各種の技法である。一般に、オペレーティングシステムでは常駐部、スワップされる部分を問わず、そのプログラムスペースを小さくすることが重要な問題である。とくに、ミニコンのオペレーティングシステムにおいては極めて重要な問題である。

第2章には文献(4)、(5)として公表したヤーノフのプログラム形の簡単化について述べている。

プログラム形 S_1 、 S_2 に同一の解釈 $\mathcal{I}$ を与え、各入力初期値について、一方が終了節点に達して停止すれば他方もそうであり、且つ、そのときの出力変数の内容（計算の最終結果）が等しいとき、 S_1 と S_2 は解釈 $\mathcal{I}$ の下で等価であるという。 S_1 と S_2 が任意の解釈の下で等価であるとき、強等価であるという。

ここでは、任意のチャートプログラムの形 S が与えられたとき、それと強等価で、節点数最小のチャートのプログラム形を一つ見つける問題を考察した。この問題がいわゆる「決定表」の問題を含んでいることを示した。このことから、この問題に対する能率のよい解法はないと考えられる。

そこで、強等価よりもさらに強い動作等価をつぎのように定義し、動作等価なものの中で節点数最小のプログラム形を求める問題を考察したが、求めるプログラム形がチャートのプログラム形の場合は、この問題は決定性有限オートマトンの状態数最小化の問題になり、最簡形を求める能率のよい手続きが知られている。第3章では、サフルーキンを用いて、さらに、スタック節点、リターン以外の節点数の総和を最小にする問題を考察した。

プログラム形 S_1 , S_2 において、それぞれの開始節点から終了節点に至る可能な道に対する演算および述語の判定結果からなる記号系列の集合が等しいとき、 S_1 と S_2 は動作等価であるという。

熟練したプログラマによって注意深く書かれたプログラムはすでに局所的にかなり最適化されており、多くの場合同じ機能に対しては同じ命令コードで書かれている。それゆえ、記号系列として等価（動作等価）な部分をまとめるということは実用的な意味を持っている。

第3章には文献(4), (6)として公表したサフルーキン化によるプログラムの単純化について述べている。

多入力サフルーキンを許したプログラム形とは、計算実行の順序を知御する機能として、ポッシュダウンスタックを用い、サフルーキンへ入るときスタック命令により戻り先番地をスタックしておき、サフルーキンの終りのリターン命令でそのときのスタックの先頭に書かれた

番地へ戻すような動作をするプログラム形である。

動作等価の定義を多入ロサフルーシンを許したプログラム形において、以下のよう拡張して使用する。多入ロサフルーシンを許した2つのプログラム形において、それぞれの開始節点から終了節点に至るすべての道に対する演算および述語の判定結果の記号系列の集合が等しいとき（スタック命令やリターン命令の実行は無視する）、この2つのプログラム形は動作等価であるという。

ここでは、任意のヤーノフのプログラム形が与えられたとき、それと動作等価で、演算節点と分岐節点の個数の和が最小（スタック節点やリターン節点を無視し）の多入ロサフルーシンを許したプログラム形を求める問題を考察し、与えられたヤーノフのプログラム形の節点数を $|V|$ とすると、その最簡形が $|V|^2 \log |V|$ に比例する手数で構成できる手続きを示した。以下にその方法を示す。

ヤーノフのプログラム形の各節点 v に対し、 v を終点とするサフルーシン可能グラフとは、その節点から終了節点へ至る道はかならず節点 v を通るような節点の集合とこれらの節点間の枝からなる部分グラフのことである。

まず、プログラム形の各節点 v に対し、 v を終点とするサフルーシン可能グラフを求める。つぎに、すべてのサフルーシン可能グラフ内で動作等価なものから一つでもあいまいかならずまとめるようにする。

なお、スタック節点、リターン節点も含めた節点数の総和が最小のプログラム形を求めるには、一カ所からしか呼ばれないサフルーシンとか小さなサフルーシンをなくすことにより、最適とは言えないが節点数の総和が少ないプログラム形が作れる。

サフルーシンなどのモジュールをもちいてプログラムを構造的に記述することは、ドキュメント、コーディング、

デバックス、改良などの立場から重要なことである。ここでは、モジュール化に対する具体的な判定基準ならびに、その判定基準に合うようにプログラムモジュール群を合併することを考える。

第4章には、文献(7)、(8)として公表したプログラムモジュール群の合併について述べている。

あるモジュール S における結果が、他のモジュール X 内でのコントロールの流れに影響を与えるとき、モジュール S からモジュール X へエフェクトがあるという、 S からエフェクトがあるモジュールの集合を S のエフェクトの範囲という。

Stevens らは、プログラム設計法の一つとして、各モジュールのエフェクトの範囲は、そのモジュール自身またはそのモジュールからつきつきとコールされて至るようなモジュールの集合（子孫のモジュールの集合）にのみ完全に含まれるように、設計の初期段階のプログラムモジュール群を再構成することを提唱している。

ここでは、エフェクトの範囲を子孫のモジュールだけに限らず、いわゆる“親”のモジュール（そのモジュールを直接コールするモジュール）と“兄弟”のモジュール（“親”のモジュールが直接コールするモジュール）にまで許すことにした。そのようにしても、“Structured design”における構造の規則性（“well formed”性）の立場から不都合さはそれほど無いであろうと思われるし、その方が合併に自由度がある。

ここでは、できるだけ多くのプログラムモジュール構成を保ちながらプログラムモジュール群を合併し、できたあとのプログラムモジュール構成において上記の条件を満たすことを考えるが、議論を簡単にするため、具体的には、できたあとのモジュールの個数が最大となるよう

な合併について考察した。

そして、コール関係のグラフ $G = (V, A)$ が有向木のとき、合併すべきモジュールの集合のすべてを $|V| + |E| \times G(|V|)$ (E はエフェクト関係を表わす枝の集合で、コールを表わす A の枝と独立に与えられる) のオーダーの手数で求められることを示した。ただし、 $G(n)$ は Hopcroft が定義したゆっくり増加する関数である。

関 連 発 表 論 文

- (1) 葛山：“PDP-11デュアルシステムにおけるリソースシェアのためのプログラムの試作”情報処理学会，ミコンのソフトウェアとネットワークシンポジウム報告集，P.78(昭和47年7月)
- (2) 本田，葛山，細見，嵩：“PDP-11/20デュアルシステムにおけるデバイスシェアについて”，昭和47年度電気関係学会関西支部連合大会 G7-28(昭和47年10月)
- (3) 本田，葛山，藤井，都倉：“PDP-11/20デュアルシステムにおけるデバイスシェアリングシステムの製作”，情報処理，14，10，P.794(昭和48年10月)
- (4) 葛山，谷口，都倉，嵩：“プログラム形の簡単化”，電子通信学会オートマトンと言語研究会資料，AL72-94(昭和48年1月)
- (5) 谷口，葛山，嵩：“ヤー/フのプログラム形の簡単化について”，電子通信学会論文誌，58-D，7，P.429(昭和50年7月)
- (6) 葛山，谷口，嵩：“サフル-4ン化によるプログラム形の簡単化”，電子通信学会論文誌D，採録決定
- (7) 葛山，谷口，嵩：“プログラムモジュール群の再構成”電子通信学会オートマトンと言語研究会資料 AL75-74(昭和51年1月)
- (8) 葛山，谷口，嵩：“プログラムモジュール群の再構成”電子通信学会論文誌D，投稿中

第1章

PDP-11/20 デュアルシステムにおけるデバイスシェアリングシステム

§ 1.1 序言

近年、複数のCPU（中央処理装置）を持ついわゆるマルチ・コンピュータ・システムが増加している。それにとどまないミニコンのネットワークに関する研究も各地で盛んに行われるようになった。⁽¹⁾ 大阪大学基礎工学部情報工学科には、2台のミニコンPDP-11/20とこれらをつなぐインタフェース装置（DA11）から成る図1.1に示すようなデュアルシステムがある。これらA・B両システムは、導入時にはそれぞれDOS（Disk Operating System）のそれぞれで独立に動作できるだけで、Aシステムからは高速紙テープリーダなどを、またBシステムからはラインプリンタなどを使用できなかった。もし、このようなデバイス（入出力装置）をA・B両システムから使えるようにすれば、各システムにデバイスを増設したのと同じ効果を得ることができる。そこで既存のDOSを改造して、コンソールタイプライタとテレタイプライタを除くすべてのデバイスを両システムから使えるようなソフトウェアシステム（デバイスシェアリングシステムと呼ぶ）を開発した。

本システムにおいては、改造および追加したプログラムはすべてモータ内部に組み込まれているため、ユーザプログラムはA・B両システムで同時に走らせることができる。各システムで独立に走っている2つのユーザプログラムが同じデバイスに同時にアクセスするとファイルの混入（例えば、ラインプリンタの出力用紙に両ユーザの出力データが交互に出力されるような事態）が起こる

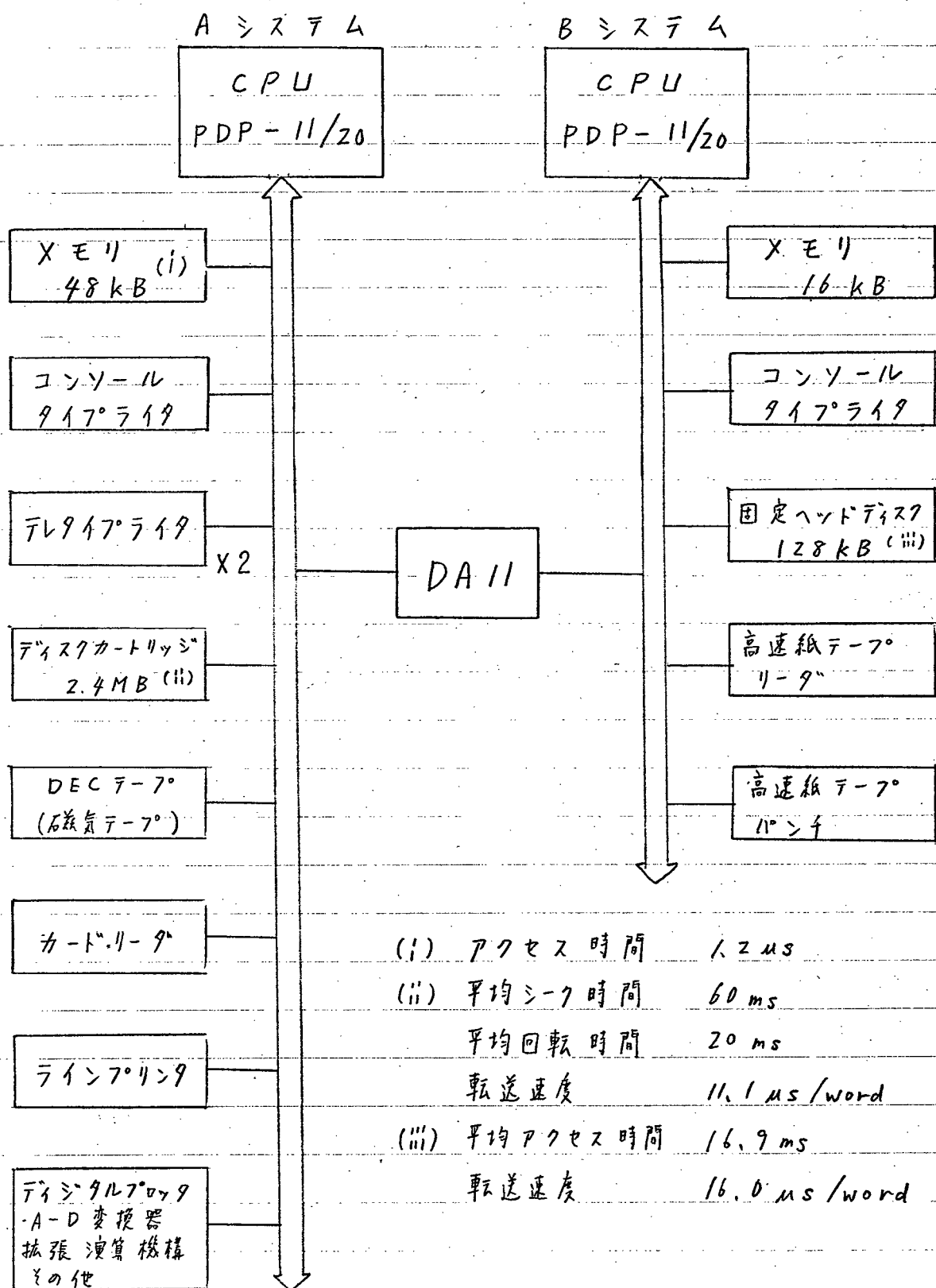


図 1-1 デュアルシステムの構成

が、既存の D O S はマルチプログラムの機能がなかったので、本システムを作製するにあたりファイルに対する相互排除が大きな問題となった。既存の D O S はインタラクティブにシステムを使用するのに適したものであるため、通常のバッチモータのようにジョブあるいはジョブステップの開始時に必要なリソースをすべて確保する方式では、デバイスを不当に長時間にわたり占有することになり、実用上デバイスシェアの効果が著しく低下する。このため、ここでは次のような相互排除の方式を取り、デバイスの確保は必要に応じて動的に行うようにした。

ファイル構造を持たないデバイス（例えば、ラインプリンタ）に対しては、ユーザプログラムからの O P E N 要求によりデバイスを占有し、C L O S E 要求によりデバイスを占有解除することにより、ファイルの混入を防止している。ファイル構造を持つデバイス（例えば、ディスクや D E C テーポ）に対しては、ユーザプログラムからの O P E N 要求によつてはデバイス内に作られたファイルを占有するだけでデバイスの全領域を占有するようなことはない。そして、ユーザプログラムからの R E A D ・ W R I T E 要求時には、デバイス制御装置を占有するだけである。このようにきめの細かい相互排除の方式を用いることによって両システムからデバイスを効率良く使用できるようにした。以下、1.2 節では P D P - 11 の概要を紹介し、1.3 節でこのシステムの作製の基本方針を、1.4 節ではここで用いたデータ転送の方法を、1.5 節では相互排除（ロック・アンロック）の問題を、また 1.6 節ではシステム全体のデバイスを管理するデバイス管理ルーチンについて述べる。

§ 1.2 既存システムの概要

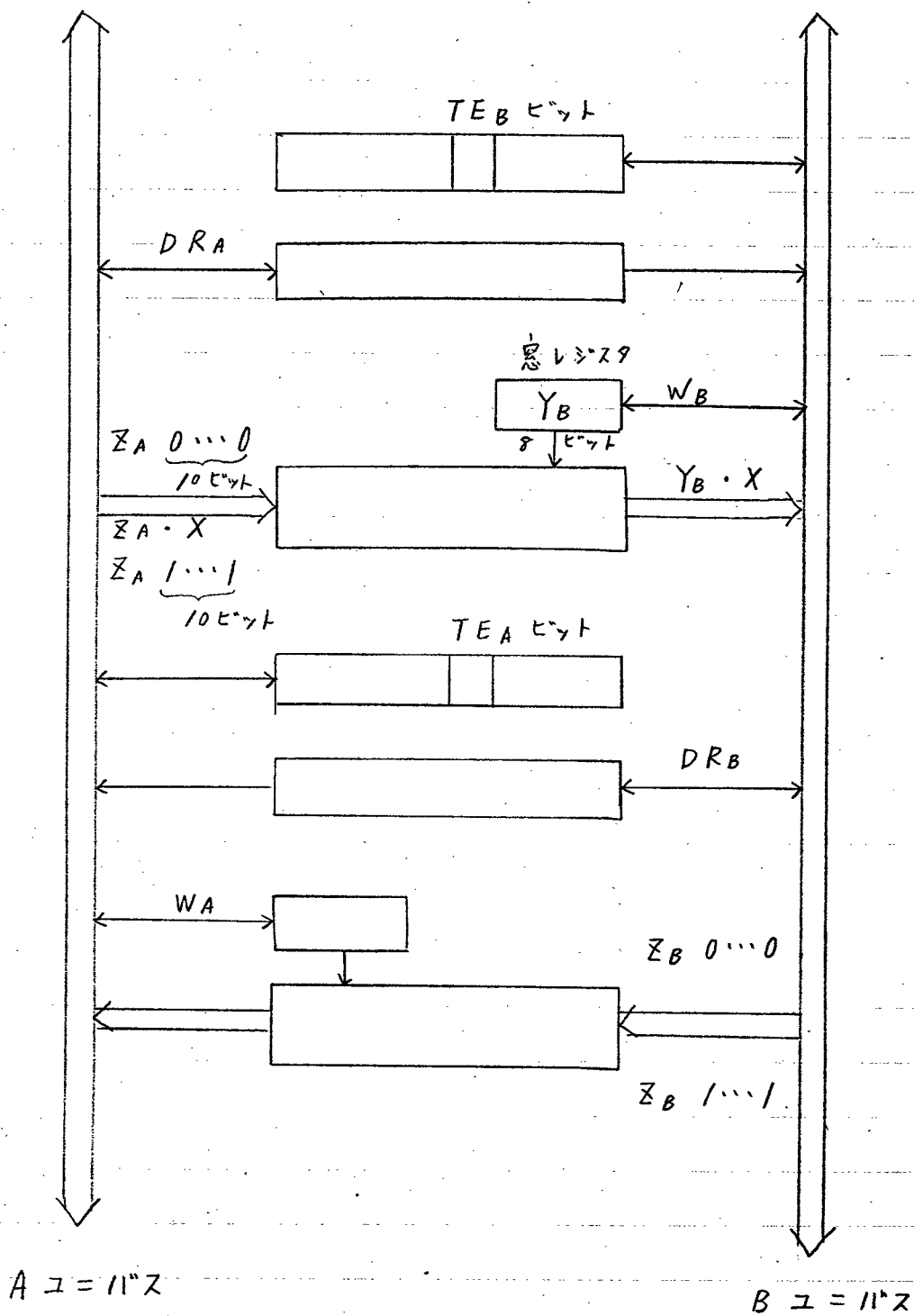
1.2.1 ハードウェア⁽²⁾

- ユニバスによる共通母線方式を用いている。
- 各デバイスの制御装置内のレジスタは、メモリと同様に番地付けられており、すべての命令は、メモリに対してと同様にこれらのレジスタに対して適用できる。
- デバイスは、それぞれ固有のメモリ・ロケーション（割込みベクトル）を指すハードウェアのポインタを持っている。このメモリ・ロケーションにデバイスに対応する割込み処理ルーチンの開始番地を設定しておくとして、デバイスからの割込み（割込みフラグと割込みベクトルがCPUに送られる）で、CPUは機械的に対応する割込み処理ルーチンを選択する。したがって、どのデバイスからの割込みかをソフト的に解析する必要はない。
- 一般的な記憶保護機構はない。
- A・B両システム間の通信を可能にするインタフェース装置DA-11を持つ。DA-11には次の2種類の通信方式がある。⁽³⁾（図1-2参照）

(i) D.M.A. (Direct Memory Access) 方式

相手のCPUを介さずに、相手の番地付けられたメモリおよびデバイスの制御装置内のレジスタに直接アクセスする方法である。今A側からユニバス上の18ビットのバスアドレスラインの上8ビットに、特定のビットパターン Z_A を下10ビットに任意のビットパターン X をのせると、DA11は Z_A を検知して、DA11中の窓レジスタ W_B の内容 Y_B を用いアドレスを $Y_B \cdot X^+$ に変換してB側のアドレスラインにのせるので、A側はB側のア

+ Y_B と X の連接 (concatenation) を意味する。



☑ 1-2

DA 11

ドレス $Y_B \cdot X$ へあたかも自分の側のアドレススペースであるかごとくアクセスでき、情報 (1 バイトまたは 2 バイト単位) を転送できる。A 側も B 側も全く対称に作られており、B 側からも同じことができる。以下いちいち断らず A 側から見て述べる。

DA 11 は A 側からと B 側からのデータ転送が同時にできるという意味で全二重の通信路と考えてよい。上のように、A で $Z_A \cdot X$ というアドレスを発すると B 側の $Y_B \cdot X$ というアドレスへ直接アクセスできるから、A から B 側の $Y_B \cdot 0 \cdots 0$ から $Y_B \cdot 1 \cdots 1$ までの 1024 バイトの領域 C_B を A 側の領域のごとくアクセスできる。もちろん B 側も C_B に直接アクセスできるから、この部分は A と B にとって B 側にある共通領域となる。

C_B の位置は窓レジスタ W_B の内容で決まる。 W_B は通常 B 側からアクセスされ、A 側から値を変えることはできないので、B 側で適当な値をセットしてもらう必要がある。一方、B 側は DA 11 のステータスレジスタの特定のビットを設定することで、A が C_B へアクセスするのを禁止できる。このように、相手側からのアクセスを制御できるという意味でシステム間の一種の記憶保護機構をとっている。

PDP-11 にはチャネル装置はないが、ディスクや DEC テープはブロック転送を行う機能を各制御装置が持っており、転送開始番地として $Z_A \cdot X$ を指定すれば、例えば、A のディスクから DA 11 を経由して B の $Y_B \cdot X$ の番地以降へ CPU A、B を問わずに転送できる。

(ii) P.T. (Program Transfer) 方式

これも全二重の方式と考えてよい。A から DA 11 中のデータレジスタ DRA (2 バイト) にデータを送ると、DA 11 は B 側に割込みをかける。B 側は、 DRA の内容を

読むことにより，Aからのデータを受けとる。

1.2.2 D O S モニタ

D O S モニタは，ユーザがコンソールを通じてファイル管理プログラム，コンパイラ，実行制御用ルーチンなどと通信を行いながら，インタラクティブにプログラムの開発実行を行うのに適したモニタであり，マルチプログラムの機能はない（図1-3にコアマップを示す）

OユーザプログラムからのI/Oマクロはモニタ中のI/Oルーチンで処理されるが，これは大きく次の2つに分れる。

(i) ロジカルI/O（略して，L I O）：データのブロック，フォーマティングを行う。ファイル構造を持つデバイス（後述）では，この他にデバイスのブロック管理も行う。

(ii) フィジカルI/O⁺（略して，P I O）：各デバイスに対して用意されており，デバイスの制御装置を直接操作し，データの転送，割込みの処理を行う。

⁺ DECではドライバと呼んでいる。

§ 1.3 システム作製の基本方針

前節 1.2.2 でお述べたように、このモニタはインタラクティブにシステムを使用するのに適したものであるため、通常のバッチモニタのようにジョブあるいはジョブステップの開始時に必要なリソースをすべて確保する方式では、デバイスを不当に長時間にわたり占有することになり、実用上デバイスシェアの効果が著しく低下する。このために、デバイスの確保は必要に際して動的に行うことが望ましい。しかし、このときファイルの混入（例えば、ラインプリンタの出力用紙に両ユーザの出力データが交互に出力されるような事態）は避けなければならない。これらを考慮して、ここでは次のような基本方針をたてた。

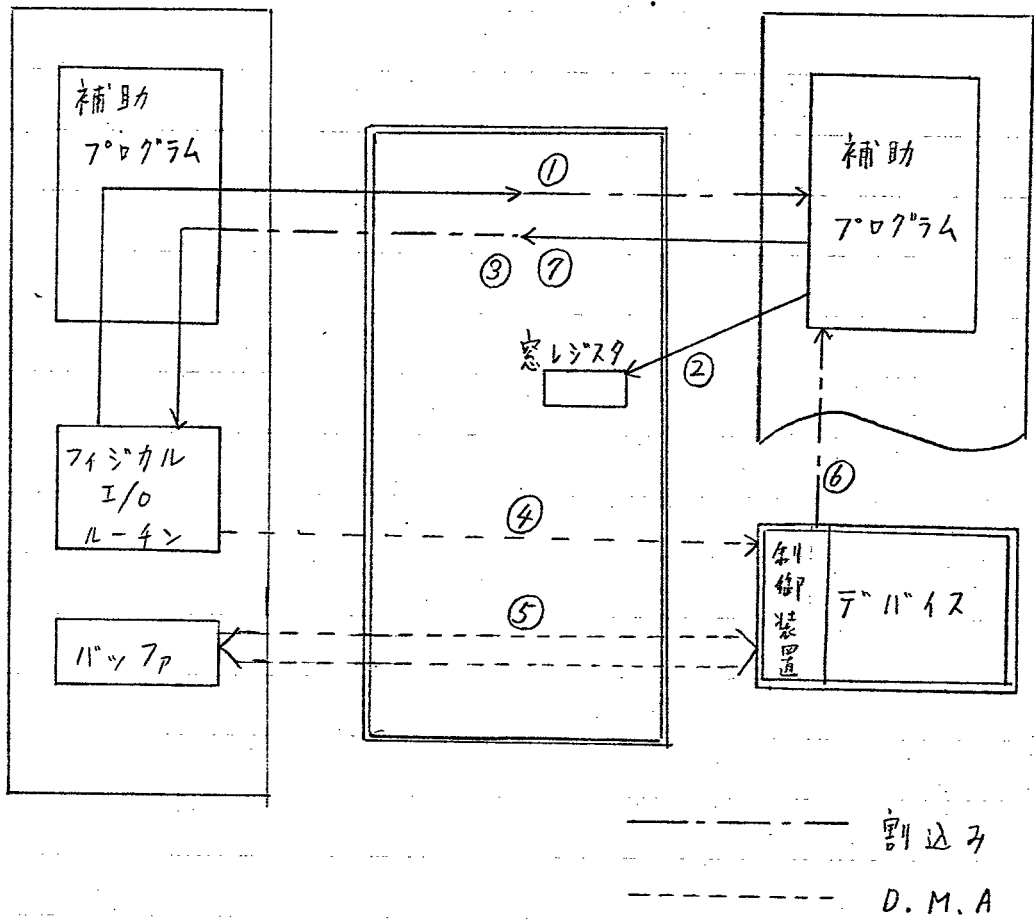
- (i) 既存のモニタの変更をできるだけ少なくする。
- (ii) 既存のシステムプログラムやユーザプログラムが新しいモニタのとおとでそのまま使えること。
- (iii) ユーザから見て、自分の使用したいデバイスがいずれの側にあるにせよ、その使用手続きが同じであること、したがってユーザがプログラムを書く際、いずれのシステムで走らせるかを考慮する必要はない。
- (iv) デバイスの占有時間をできるだけ短くすること。

§ 1.4 データ転送

以上のような現有システムの特徴および基本方針からデータ転送は次のような方法で行うことにした。

使用するデバイスがどちら側のものであっても使用する側の P I O が直接 (デバイスが相手側であれば, D A 11 を通して D. M. A 方式で) デバイスを制御する。相手側のデバイスの場合, 相手側の C P U がこの I/O 操作に対して補助することは, 使用する側が直接デバイスを制御できるように D A 11 中の窓レジスタを設定することと, そのデバイスからの割込みを相手側 (そのデバイスを使用している側) に伝達することだけである。したがって, 相手側デバイスに対するデータ転送要求ならびにデバイス制御装置によるデータ転送は図 1.4 のようになる。

図 1.4 の補助プログラム (後述のデバイス管理ルーチン) のうち, 相手側のものはこの I/O 操作に対する前述の補助を, また自分の側のものは I/O ルーチンと相手側の補助プログラムとの間の通信の補助を行う。なお, 自分の側のデバイスに対しては, 従来どおりの ④ ⑤ ⑥ の手続きで行われる。ただし, この他にデバイス確保の手続き (後述) も必要である。



- ① レジスタの設定依頼 (P.T.方式)
- ② レジスタの設定
- ③ レジスタ設定完了の通知 (P.T.方式)
- ④ デバイス制御レジスタの設定 (D.M.A.方式)
- ⑤ データ転送 (D.M.A方式)
- ⑥ デバイスからの割り込み
- ⑦ 割り込み伝達 (P.T.方式)

図 1-4 相手システムとのデバイスとのデータ転送

§ 1.5 デバイスのロック・アンロック

I/O 要求は、まず LIO で処理され、LIO から PIO にデバイスの操作要求が出される（モータ自身からのものは直接 PIO に要求が出されることがある）。前節で述べたようなデータ転送の方法では、デバイスを直接制御する PIO が、各デバイスにつけて、両システムに1つずつあることになるから、同時に同じデバイスにアクセスしようとすることも起こり得る。またデバイス（ラインプリンタのような）によっては、交互に同じデバイスにアクセスするとファイルの混入を起す可能性がある。このような事態を避けるためには、例えば A システムがあるデバイスを使う場合には、安全にデータ転送を行うのに必要な期間 B システムがこのデバイスを使うことを禁止すればよい。このように、あるデバイスを一定期間他のシステムに使わせないようにすることをそのデバイスをロックするという。ロックしたデバイスに対して、この禁止を解くことをアンロックするという。デバイスシェアリングシステムでは、全システムのデバイスを管理するデバイス管理ルーチン（DEMON-DEVICE MONiter-と略す）が用意されており、この DEMON に対して、I/O ルーチンが必要な時点で、必要なデバイスのロック・アンロックの要求を出すことにより、これが実現される。DEMON におけるデバイス管理の手法は 1.6 節にゆずるとして、ここでは、I/O ルーチンがどのようにしてロックをかけることにより、ファイルの混入などの問題を解決しているかを述べる。

1.5.1 ロック方式

各デバイスのI/Oルーチンは、その性格により、次の3つのロック方式のうちのいくつかを組み合わせて用いる。

○転送ロック：PIOがフィジカルな1レコードのデータを転送するために、そのデバイスの制御装置（および必要ならDA11）を占有するためのロックである。PIOはDEMONに対して、そのデバイスの転送ロック要求を出し、転送許可の応答が来ればロックされたとみなす。このとき、相手システムがすでにこのデバイスをロックしていたならば、この要求は、DEMON中で待ち行列に入れられ、アンロックされた時点で転送許可の応答が出る。アンロックは、転送ロック解除の要求をDEMONに発することにより行われる。

○占有ロック：LIOがファイルの混入防止のためにかけるロックである。DEMONにそのデバイスの占有要求を出し、占有許可ができればロックがかかったことを意味する。ラインプリンタのようなデバイスを占有したいがすでに他システムにより占有ロックされている場合には、待ち行列には入らず、占有不能がI/Oルーチンを経てオペレータに伝えられる。

○ビットマップロック：1.5.3で述べる

ここで、ラインプリンタのようなデバイスの占有ロックに対して待ち行列が用意されていないのは、例えば次のような事態（デッドロック）をさけるためである。

Aシステムがあるデバイスaを先ず占有し、次にBシステムが別のデバイスbを占有する。さらにAシステムがデバイスbを、Bシステムがデバイスaを占有しようとする。もし、占有ロックに対して待ち行列を用意すると、これらの要求は待ち行列に入るが、両システムともデバイスa、bがそろわないと処理ができなるとすれ

ば、デバイス a , b は占有ロックが解除されることなく永久に処理できない。

一方、デバイスを占有できないときはその旨をオペレータに知らせる方式では、オペレータはデバイスがいたことを確認してから続行指令を発すればよい。したがってデッドロックが生じた場合には、両システムのコンソールに占有不能のメッセージが出力されることになり、デッドロックが検出できる。転送ロックでは、ロックされたデバイスはいつかその処理が終り必ずアンロックされるので、待ち行列を設けてもデッドロックは生じない。

使用頻度の高い $DA11$ は、転送ロックのみが許され、占有ロックは許されていない。

各デバイスの I/O ルーチンが、どのロック方式をどのように組み合わせて用いるかは、占有時間の短縮およびファイルの混入の問題を考慮して決定する必要がある。ここでは、デバイスをファイル構造を持つものとそうでないものとに大別して考える。

1.5.2 ファイル構造を持たないデバイス

ラインプリンタやカードリーダーのようなファイル構造を持たないデバイスでは、ファイルの混入を避けるため、ユーザからの $OPEN$ マクロ受付け時にデバイスの占有ロックを行う。引き続き I/O 要求は転送ロックを必要とする。これは、その I/O 処理に $DA11$ を必要とする場合、占有ロックできない $DA11$ を転送ロックするためである。ユーザからの $CLOSE$ マクロ受付け時に、この占有ロックは解除される。図 1.5 には、両システムからこの種の同じデバイスに同時に I/O 処理を始めようとした場合の流れを示す。(以下の図では、 $DEMON$ が全システ

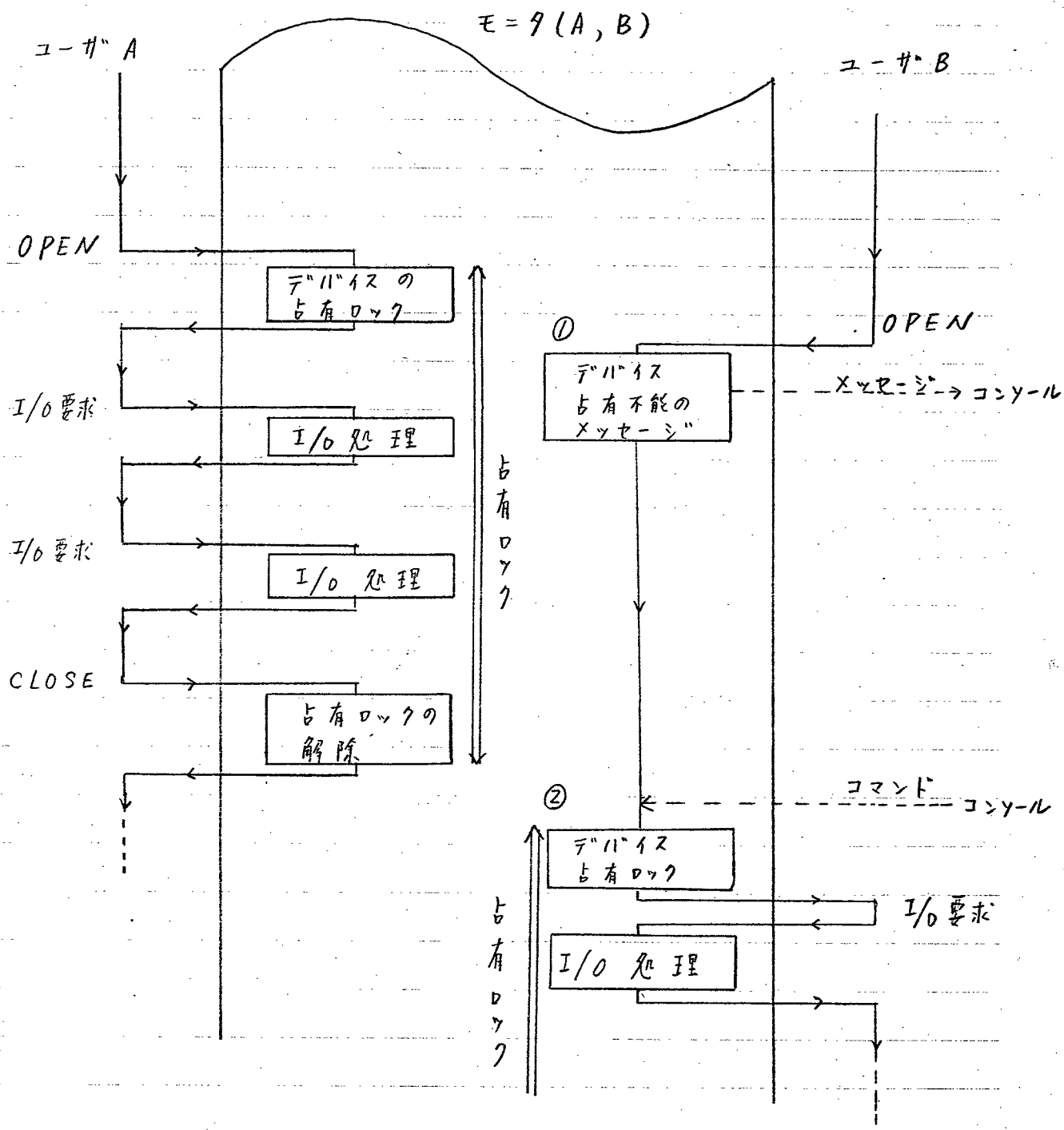


図 1-5 ラインプリンタカードリーダのロック・アンロック

ムのデバイスを管理していることを考慮して、図の簡単のため両モータを一つにまとめて書く。) ①では、OPENマクロ受け時に占有ロックできなかったため占有不能をコンソールに出カし、オペレータの指示を待っている。②でオペレータが、そのデバイスがあったことを確認して続行指令を発すと、処理を続行する。

1.5.3 ファイル構造を持つデバイス

ディスクやDECテープのようなファイル構造を持つデバイスにはディレクトリーと呼ばれる登録表があり、これにより複数のファイルを管理している。ファイルは複数のブロックより構成されているため、ブロックを資源とみなせる。各ブロックが空いているかどうかの情報は、そのデバイス中のPBM(パーマネントビットマップ)に集中的に保持されている。PBMは一般に数ページにわかれている。

入出力とるOPEN時にそのファイルをロックしCLOSE時にアンロックする機能は、すでにDOSに含まれているため、書き込み中のファイルを誤って読むことはない。そのため、このようなデバイスからのファイル読み込みは転送ロックのみで充分である。

一方、データの書き込みに関しては少し事情が異なる。データをこれらのデバイスに書き込む場合には、ユーザが発する最初のOPENO(データの集まりを新しく作るファイルに出カするための)マクロ、あるいはOPENE(すであるファイルのうしろにデータを追加するための)マクロにより、モータはPBMの適当な1ページをメモリに読み込み(これをCBM-コアビットマップという)、これを用いて空きブロックを探し、この空き

ブロックにデータを書き込み同時にCBMを更新する。引き続き出力要求に対してはこのCBMが用いられ、このCBMに空きブロックが無くなれば、これをデバイス上に返し別のPBMを読み込む。また、CBMはユーザが発する最後のCLOSEマクロによってもデバイス上に返される。

したがって、両システムから同じデバイスに出力要求を出すと、同じページのPBMを読み込む可能性があり、この場合同じブロックに書き込む事態が起る。このような事態を防ぎ、またロック時間の長い占有ロックをできるだけ避けるため、ここでは転送ロック、占有ロックの他にビットマップロックを用いる。これはPBMをメモリ上に読み込んだ段階でそのPBMの特定の場所の使用中であることを示すビットをセットし、他のシステムがPBMのこのページを使うことを禁止することである。したがって、このページのPBMが代表するブロック群を占有ロックしたのと同じ効果を得る。これにより、デバイスの占有ロックを行わなくても、同一ブロックに両システムから書き込むことはなくなる。PBMを読み込みそのページをロックするまでの間はデバイスの占有ロックを行う。PBMのアンロックはCBMをPBMに戻す際に行われる。このようなロック方式においては、ロックされたPBMを読み込んだ場合には、ロックされていない他のPBMを読み込んで使用すればよい。このときの流れを図1-6に示す。①ではまずPBMの1ページ目を読むがロックされているため、2ページ目を読み、これにロックをかけて以後で使用することを示している。

以上のようなロック方式により、このようなデバイスに対するロック時間が非常に短縮される。これは、使用頻度上からも非常に大きな利点である。

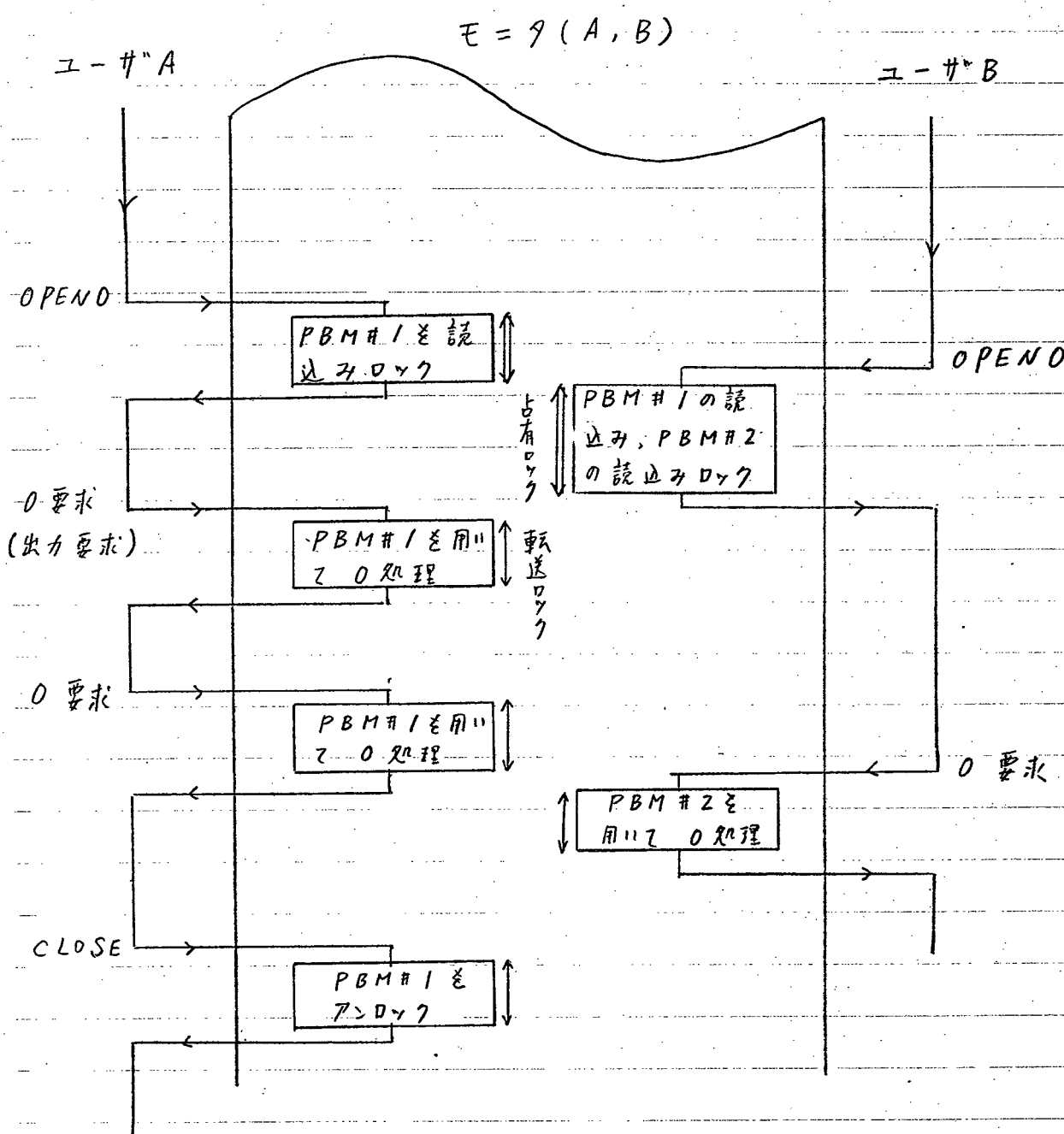


図 1-6 ティースのロック・アンロック

§ 1.6 デバイス管理ルーチン (DEMON)

DEMON は、相手側のデバイス操作に対する 1.4 節に述べたような補助および 1.5 節で述べたロック・アンロックの要求に対する処理を行い、全デバイスを管理するルーチンである。ただし、システム固有のものとして扱うのが望ましいコンソールや、フィジカル I/O を持たないテレタイプは管理の対象になっておらず、また DA は一種のリソースとして DEMON が管理する。

DEMON は両システムのモータの常駐部に組み込まれ、それぞれは自分の側のデバイスを管理し、また互いに交信しながら全デバイスを管理する。各システムの DEMON は全く対称に作られており、主従の関係はない。そこで以下特に断らない限り A システムを中心とし、B システムを相手側システムとして説明する。

1.6.1 DEMON の機能

(1) デバイス管理テーブル

1.5 節でも述べたように、モータ中の LIO や PIO は、デバイスを転送ロック、占有ロックするために、転送ロック要求、転送ロック解除、占有ロック要求、占有ロック解除を必要に応じて DEMON に発する。これらの要求は、そのデバイスがいずれの側にあるにせよ、自分の側の DEMON に対して出され、DEMON は、そのデバイスが自分の側のものであるばここで調べ、相手側のデバイスであれば相手側の DEMON にロックの可否をたずね、要求を発した I/O ルーチンに返答する。

A システムの DEMON は、A 側のデバイスを図 1-7 に示す種類のテーブルで管理している。各テーブルは

A側のための領域とB側のための領域にわかれており、図1-7の各領域のビット位置がA側の各デバイスと1対1に対応している。

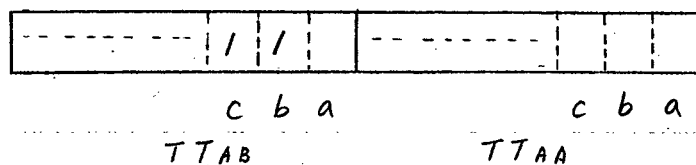
○ 転送テーブル (TTA: TTAA および TTAB)

A側の各デバイスが現在どちらのシステムにより転送ロックされているかを示す。図1-7では、BシステムがA側のデバイスb, cを転送ロックしている。

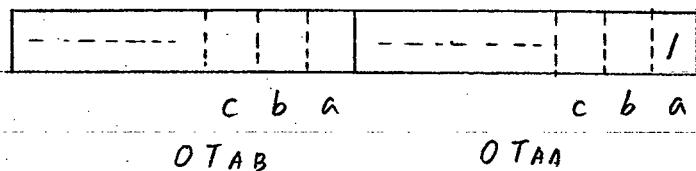
○ 占有テーブル (OTA: OTAA および OTAB)

A側の各デバイスが、現在どちらのシステムにより占有ロックされているかを示す。図1-7では、AシステムがA側のデバイスaを占有ロックしている。

転送テーブル TTA



占有テーブル OTA



予約テーブル RTA

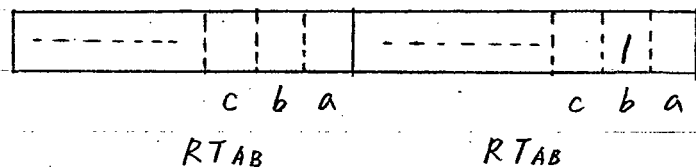


図1-7 デバイス管理テーブル

○ 予約テーブル (RTA : $RTAA$ および $RTAB$, STA)
このテーブルはファイル構造を持つ デバイスに対して用意された一種の待ち行列である (ただし, 行列の長さは 1 以下)。図 1-7 には, 転送ロック要求を予約するためのテーブル RTA のみしか書かれていないが, 占有ロック要求 (1.5 節で述べたように, PBM を読み込みそのページをロックするまでの間はデバイスを占有ロックする) のための予約テーブル STA がある。以下では, 転送ロックについて述べる

すでに一方のシステムによって転送あるいは占有ロックされている A 側のデバイスに対して, 他方のシステムから転送ロック要求が出されたとき, このテーブルがセットされる (1.5.1 節で述べた “待ち行列に入れる” に対応している)。図 1-7 では, A システムからデバイス b の転送ロック要求が出されたが, B システムがすでに転送ロックを行なっているため, $RTAA$ の対応するビットがセットされている。

(11) ファイル構造を持つデバイスに対する転送ロック要求, 転送ロック解除の処理

図 1-7 に示したようなテーブルを用いて, A 側の DEM は, A 側のファイル構造を持つデバイスに対する転送ロック要求, 転送ロック解除を図 1-8, 図 1-9 にそって処理する。ここで,  は A 側の I/O ルーチンからの要求に対する処理の入リ口であり,  は B 側の DEM からの割込みに対する処理の入リ口である。また, 破線で囲まれた部分では, 優先度を上げることによりすべての割込みを禁止している。以下に, デバイス (1) に対し, 転送ロック要求および転送ロック解除が例 1, 例

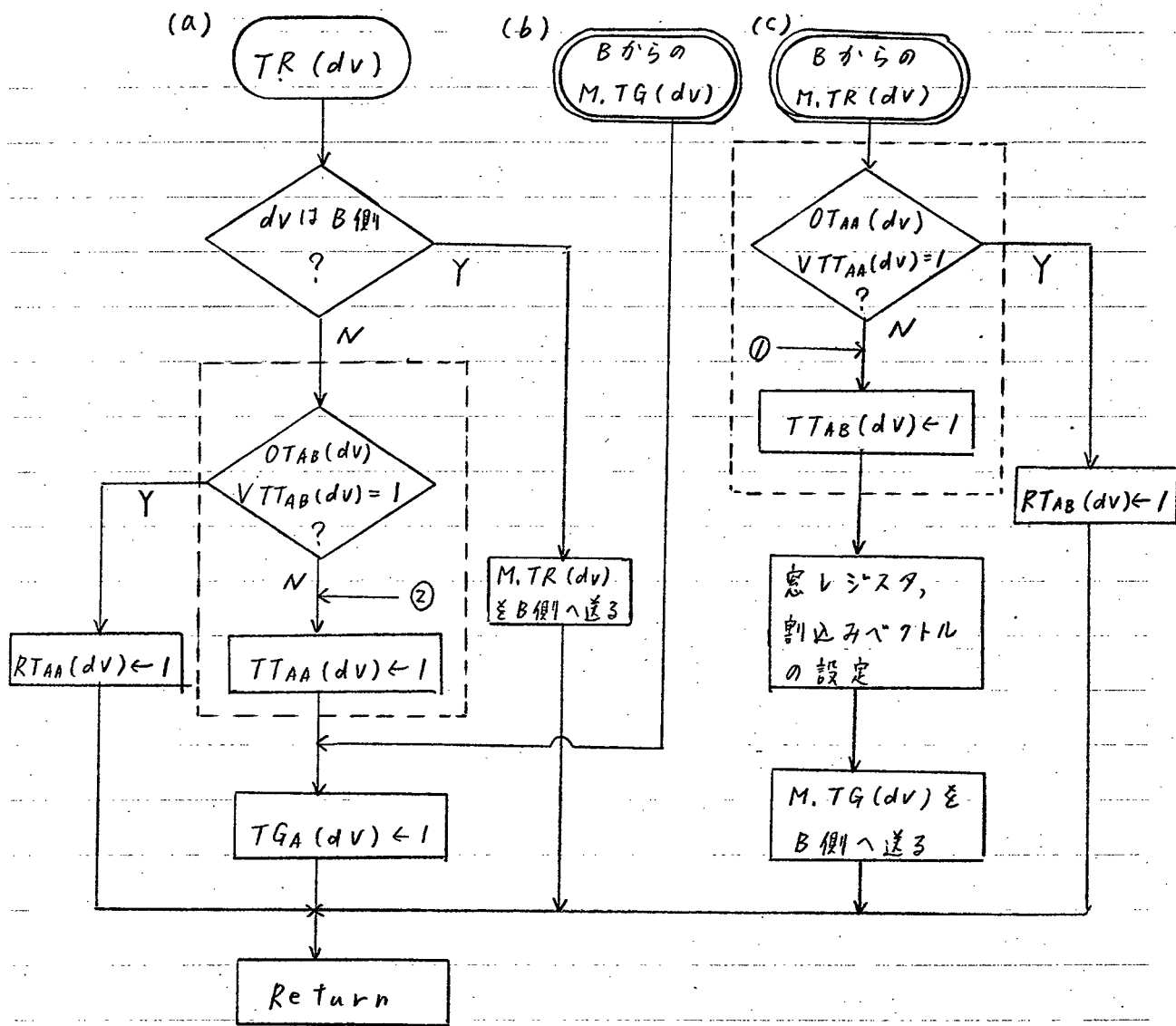


図 1-8 A 側 DEMON の転送ロック要求の処理

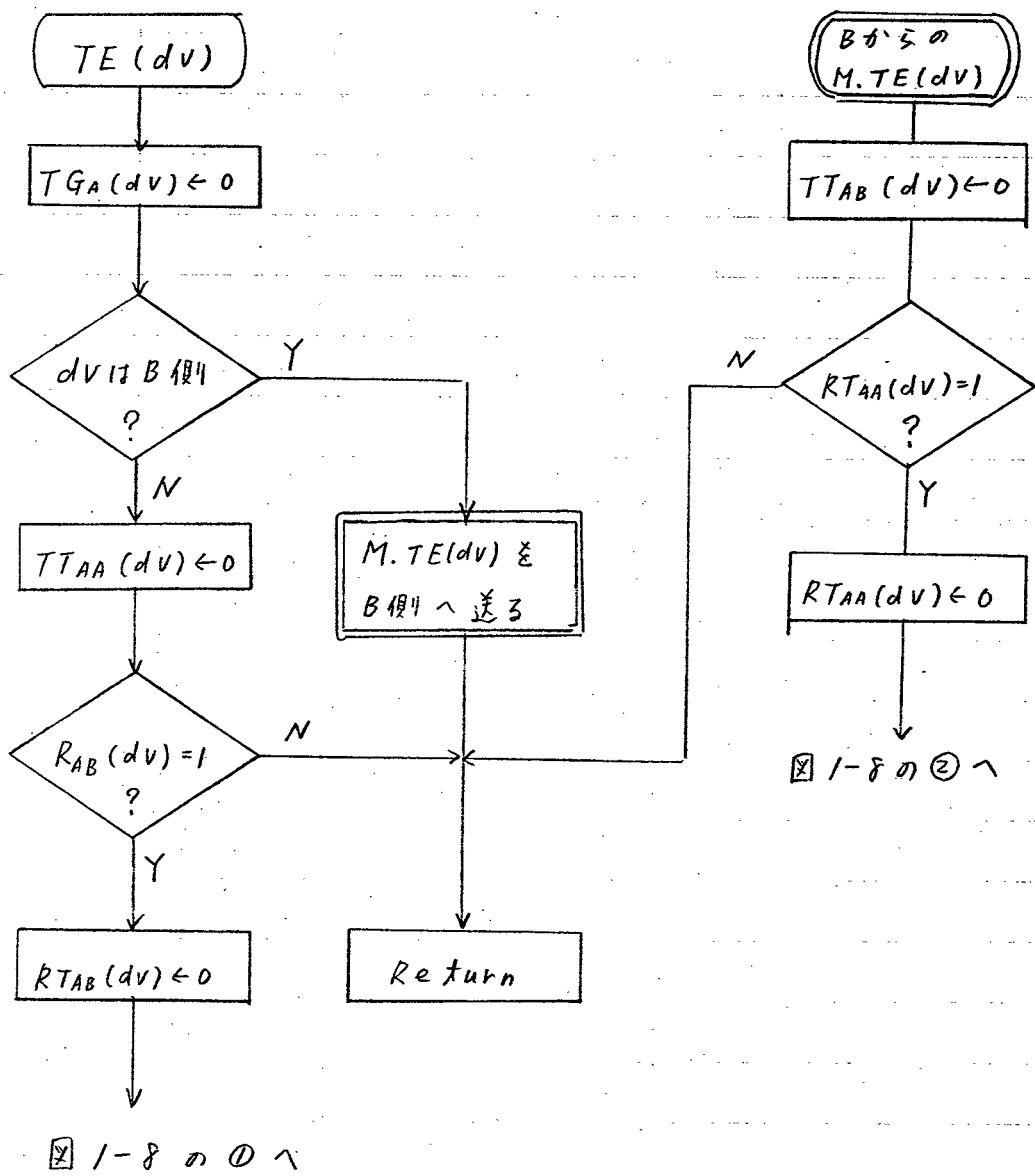


図 1-9 A 側 DEMON の転送ロック解除の処理

2, 例3, 例4の順に起こった場合について説明する

○例1 AシステムからAシステムのデバイス(1)の転送ロック要求

P I O からデバイス名(1)をパラメータとする転送ロック要求 T R (1) は, 図 1-8 (a) の部分に入る. 各デバイスがいずれの側にあるのかを示すテーブル (図 1-7 には示されていない) により, このデバイスが A 側にあることを知る. そこで, B 側がこのデバイスをロックしていないかどうかを調べ, ロックしていなければ T T A A (1) および P I O との通信領域である T G A (1) をセットし, 転送ロック許可を知らせる. P I O は, コントロールが戻ると常に T G A (1) を見ており, T G A (1) がセットされればデバイス(1)を使い始める.

○例2 BシステムからAシステムのデバイス(1)の転送ロック要求

この要求は, B 側 D E M O N の図 1-8 (a) の部分に入る. デバイスは A 側のものであるから, A 側の D E M O N に処理を依頼する (M. T R (1)). これは割込みによって A 側の図 1-8 (c) に入り, ここでこのデバイスを A 側がロックしているかどうかを調べる. ここでは例 1 によってデバイス(1)がロックされているため, B のための予約テーブル R T A B (1) をセットして割込みから戻る. B 側では処理を依頼した後すぐに P I O に戻るが, B 側の通信領域 T G B (1) がセットされていないため, デバイス(1)を使用できない.

○例3 AシステムからAシステムのデバイス(1)の転送ロック解除

この要求は図 1-9 (a) の部分に入る. 通信領域 T G A (1) をクリアし, A 側のデバイスであるから T T A A (1) をクリアする. 次に, このデバイスに対する B 側の待ちを調べ

る。例2で $RTAB(1)$ をセットされているため、図1-8の①へとび、転送テーブル $T TAB(1)$ をセットし、 DA の窓レジスタに、 B システムから DA を通してデバイス(1)の制御レジスタに D, M, A 方式でアクセスできるように値を設定する。同時に、そのデバイスの割込みベクトルに $DEMON$ の割込み伝達ルーチンの入り口の番地を設定する。その後、 B 側の $DEMON$ にデバイス(1)の転送ロック許可を応答する ($M, TG(1)$)。この応答は B 側の $DEMON$ の図1-8(b)の部分に伝わり、通信領域 $TG_B(1)$ をセットする。これで、例2の要求が許可されたわけで、この段階で B システムはデバイス(1)を使用できる。

○例4 B システムから A システムのデバイス(1)の転送ロック解除

この要求は、 B 側 $DEMON$ の図1-9(a)の部分に入る。まず通信領域 $TG_B(1)$ をクリアし、 A 側のデバイスであるため、 A 側へ $M, TE(1)$ を送る。これは A 側の図1-9(b)の部分に入り、 $T TAB(1)$ をクリアし、このデバイスに対する A 側の待ちを調べる。待ちがなければただちに戻り、待ちがあれば $RTAA(1)$ をクリアし図1-8の②の部分へとび、以下、例1と同様の処理を続ける。

(iii) ファイル構造を持つデバイスに対する占有ロック要求や占有ロック解除も上の(ii)とほぼ同様の過程で処理される。

(iv) ファイル構造を持たないデバイスに対しては、転送、あるいは、占有ロック要求に対する待ち行列(予約テーブル)が用意されていることは、前にも述べたとおりである。

(V) その他の D E M O N の機能

例3において、デバイス(1)からの割込みはA側にかかるので、これをB側に伝える必要がある。デバイス(1)からの割込みは、割込みベクトルを介して D E M O N 中の割込み伝達ルーチンに入り、割込み伝達ルーチンは、デバイス名をパラメータとしてB側へ割込み伝達メッセージを送る。B側の D E M O N 中の割込み伝達受理ルーチンが、割込みによってこれを受け、そのデバイスの割込み処理ルーチンにコントロールを移し、割込み処理を任せる。

D E M O N には、この他に、リセット命令(そのシステム側のすべてのデバイスのI/O処理を中断させる命令)の処理の機能もある。

5.1.7 結言

デバイス管理ルーチンの作成およびフィジカルI/Oルーチン、ロジカルI/Oルーチンの変更は、すべてアセンブリ言語で行なった。デバイス管理ルーチンの大きさは約300ステップ/1000バイトであり、それ以外はオーバレイ部で処理されるため、モータの常駐部の増加分は約1000バイトである。それゆえ、本システムを動かすために必要な主記憶の大きさは、既存のDOSが動く最小値の8Kバイトにとどめることができた。

このシステムを作製するために既存のDOSを解説したが、それに約4カ人月費やした。BシステムへのDOSのシステムジェネレーションならびにこのシステムの基本設計に約2カ人月費やした。また、このシステムの詳細設計ならびにプログラムの作製に約4カ人月費やした。

なお、本システムは1972年12月に動き始め、以降実用に供せられている。ミニコンピュータでのこの種の試みの中では早い時期に実現されたものの一つである。

第2章

ヤーノフのプログラム形の簡単化

§ 2.1 序言

筆者は第1章で述べたデバイスシミュリングシステムを制作するためミニコンピュータ PDP-11/20 のデバイスオペレーティングシステム DOS を分析したが、プログラミング技法として注目されたことはプログラムの長さをできるだけ短かくするための各種の技法である。一般に、オペレーティングシステムでは常駐部、スロップされる部分を問わず、そのプログラムスペースを小さくすることは極めて重要な問題の一つである。ここでは、プログラムが扱うデータセットはただ一つのレジスタに格納されているものとし、データセットに対する各種命令はそのレジスタに対する命令と考えて、いわゆるデータセットの構造そのものは考えないで、単にプログラムの構造 — プログラム形 — について考える。そして、有向グラフで表現されたプログラム形が与えられたとき、終了節点までの部分グラフで等価なものがあれば、それらを一つの部分グラフにまとめることにより節点数の総和を最小にする問題について考察する。

ヤーノフのプログラム形とは、有限状態のコントローラとただ一つのレジスタ X (ただし、無限の情報が入りうる) をもつ機械に対する抽象的なプログラムであり、有限オートマトンの状態遷移図のように表わされる。開始節点と終了節点を除く各節点は関数記号をラベルとしてもつ演算節点かまたはそこから出る枝に述語記号をラベルとしてもつ分岐節点のいずれかである。プログラム形 S にある解釈 I を与えるとは、各関数記号、述語記号に具体的なある関数、述語を割り当てることであり、こ

れで一つのプログラム $\langle S, \rho \rangle$ になる。プログラム形 S_1, S_2 が解釈 ρ の下で等価であるとはプログラム $\langle S_1, \rho \rangle, \langle S_2, \rho \rangle$ が各入力初期値について、一方が終了節点に達して停止すれば他の方もそうであり、かつ、そのときの出力変数が等しいことである。 S_1, S_2 が任意の解釈 ρ の下で等価であるとき、 S_1, S_2 は強等価であるという。

この章では、任意のヤーフのプログラム形 S が与えられたとき、それと強等価で、演算節点、分岐節点の節点数の総和が最小であるようなヤーフのプログラム形 S_m を見つける問題を考察する。ヤーフのプログラム形において、演算のあとには必ず現われているすべての述語に関する判定を行なうようにし、演算とこの“複合述語”とをあわせて一つの節点とみなしたものを準プログラム形という。準プログラム形の単純化は、完全に指定された決定性有限オートマトンの単純化の問題と同じである⁽⁶⁾。この章では、求めるヤーフのプログラム形の最簡形の演算節点の個数は、それと等価な準プログラム形の最簡形の節点の個数に等しいことを示し、まず、演算節点数を準プログラム形において最小化し、つぎに、もとのヤーフのプログラム形にもとずくとき、分岐節点数を最小にすることを考える。しかし、この分岐節点を最小にする問題は、いわゆる“決定表”の問題であり、現在の所、能率のよい手続きは知られていない。

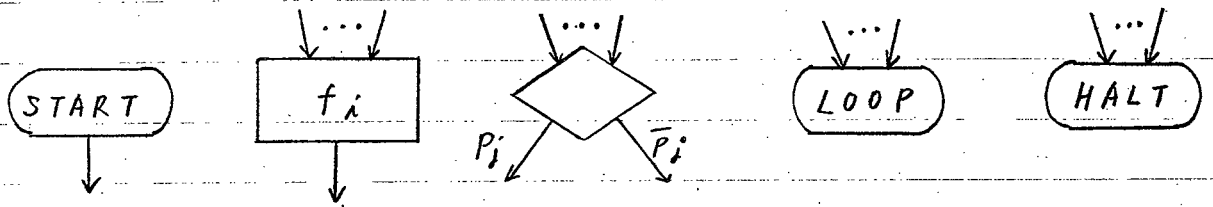
§ 2.2 定義と問題

(一変数の)関数記号の集合を F , 述語記号の集合を P とする。(F, P 上の) ヤーノフのプログラム形 S とは、その命令が図 2-1 に示されるような形をしたプログラム形をいう⁽⁷⁾。ただし、開始節点はただ一つである。演算節点はある関数記号 $f_i \in F$ をラベルとしてもち、分岐節点から出る二本の枝はある述語記号 $P_j \in P$ およびその否定形 $\bar{P}_j$ をラベルとしてとつ。

このプログラム形 S に現われている関数記号および述語記号にいわゆる“解釈”を与えと、いわゆるプログラム $\langle S, \mathcal{I} \rangle$ になり。 $\langle S, \mathcal{I} \rangle$ の開始節点において内部変数にある初期値を代入すると、“計算”が実行される⁽⁷⁾。演算節点では、ラベルとして書かれた関数記号 f_i (に与えられた“解釈”)が示す演算を実行する。ラベル P_j および $\bar{P}_j$ の二本の出力枝をとつ分岐節点では P_j (に与えられた“解釈”)が示す述語判定を行ない、 T (真を表わす) ならラベル P_j の枝を選び、 F (偽を表わす) ならラベル $\bar{P}_j$ の枝を選び、計算の制御をつぎの節点に移す。ループ節点は無限ループを表わし、この節点からはどこへも制御が移らない。そして、終了節点では変数の内容を出カして停止する。

プログラム形 S_1, S_2 が解釈 $\mathcal{I}$ の下で等価であるとはプログラム $\langle S_1, \mathcal{I} \rangle, \langle S_2, \mathcal{I} \rangle$ が各入力初期値について、一方が終了節点に達して停止すれば他の方ともそうであり、かつ、そのときの出力変数の内容(計算の最終結果)が等しいことをいう。 S_1, S_2 が任意の解釈 $\mathcal{I}$ について等価であるとき、強等価であるといい、 $S_1 \equiv S_2$ と書く。

この章では以下に示す問題 2.1 を考察する。



開始節点

演算節点

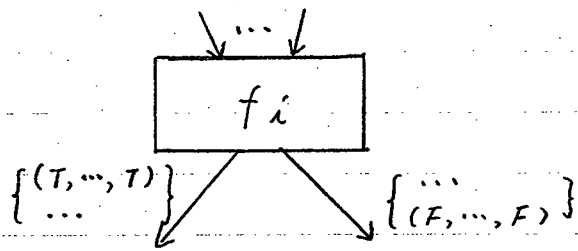
分岐節点

ループ節点

終了節点

図 2-1

ヤ-17 のプログラム形の命令



準プログラム形の演算節点

図 2-2

準プログラム形における命令

[問題 2.1] 任意のヤーノフのプログラム形 S が与えられたとき，それと強等価で，節点数最小のヤーノフのプログラム形 $S_{\min}$ (これを最簡形と呼ぶ) を (一つ) 見つけよ。

我々はその手段として，次のような“準プログラム形”を利用する。準プログラム形とは，関数記号による演算後，現れているすべての述語記号に関する判定を行ない，その演算とこの“複合述語”による判定とを合わせて一つの節点とみなしたものである。⁽⁶⁾ この節点も単に演算節点と呼ぶ，述語記号に順をつけ $P_1, \dots, P_m$ とおくと，演算節点から出る枝には $\{T, F\}^m$ の部分集合 L をラベルとしてつける (図 2-2)。 $P_1, \dots, P_m$ による判定結果が $a_1, \dots, a_m$ (各 a_i は T か F) であれば， $(a_1, \dots, a_m)$ を含む L をもつ枝を選んで計算が実行される⁺。同一の節点から出る枝については，ラベルは互いに背反 (共通要素をもたない) であり，また， $\{T, F\}^m$ の任意の元はよのどれかの枝のラベルに含まれているとする。

さて，与えられた任意のヤーノフのプログラム形を S とし，求める S の最簡形 (の一つ) を $S_{\min}$ とする。 S と強等価な⁺⁺ 準プログラム形の最簡形を $\bar{S}_{\min}$ とする。明らかに， $S_{\min}$, $\bar{S}_{\min}$ はただ一つの終了節点ともち，ループ節点も，もしあれば，ただ一つである。

⁺ 開始節点の直後に述語による判定を行う場合は，判定のみの節点を一つ設け，これを例外的に扱えばよい。

⁺⁺ 準プログラム形に対しても，同様に，強等価性を定義する。

§ 2.3 最簡形 S_{min} を求める手順

S_{min} を求める手順を示す前に、まず、のちに述べる手順で S_{min} を求めることができることを示す定理を述べる。

[定理 2.1] $\bar{S}_{min}$ の節点の集合を $\bar{V}$ 、 S_{min} の判定節点を除いた節点の集合を V とすると、 $\bar{V}$ と V の要素数は等しく $\bar{V}$ から V への 1 対 1 の対応 P で次の条件を満たすものが存在する。

(a) 開始節点、終了節点、ループ節点 (もしあれば) は互いに対応する

(b) $\bar{S}_{min}$ の演算節点 (すなわち、開始、終了、ループ節点以外の節点) $\bar{v}$ と S_{min} の演算節点 $P(\bar{v})$ で演算される関数記号は同じである。

(c) $\bar{S}_{min}$ の節点 $\bar{v}_1, \bar{v}_2$ 間の枝のラベルを L ($L \in \{T, F\}^m$) とする。一方、 S_{min} の節点 $P(\bar{v}_1)$ から $P(\bar{v}_2)$ へ (途中、演算節点を通過することなく) 計算の実行の制御をうつすような、述語記号 $P_1, \dots, P_m$ による判定結果 ($a_1, \dots, a_m$) (各 a_i は T か F) の集合を L' とすると、 $L = L'$ である。

この定理は、 $\bar{S}_{min}$ の演算節点と S_{min} の演算節点を同一視でき、その間の各述語の判定結果も部分的に見て対応しているという強い結果を述べている。

この定理により、与えられた S から最簡形 S_{min} を求めるには、(i) S をそれと強等価な準プログラム形に変更し、(ii) 準プログラム形における最簡形 $\bar{S}_{min}$ を求め、(iii) その節点を S_{min} の (判定節点以外の) 節点としてそのまま採用し (演算節点の関数記号は同じもの)、その後、(iv) S_{min} の開始、終了、ループ節点も含め、各演算節点間を、 $\bar{S}_{min}$ の元の枝のラベルに含まれていた判定結果と

矛盾しないように、全体で最小となるよう、判定節点を
うめこめばよい。

問題 (i), (ii) については、その手続きはよく知られて
おり^{(6)~(8)}、困難さはない。特に、(ii) は決定性有限
オートマトンの状態数最小化の問題と同じであり^{(6), (7)}、
能率のよいアルゴリズムが知られている⁽⁸⁾。従って、 S_{min}
を求める問題は (iv) に帰着される。この問題は、いわゆ
る“決定表”の問題⁽⁹⁾を含んでおり、組合せ論的性格を
まぬがれず、現在のところ能率のよい方法は知られてい
ない。

S_{min} の任意の節点 v に対し, S_{min} の開始節点から v まで出ていた枝を v へ入るようにつなぎなおしてできるプログラム形を $S_{min}(v)$ と書く. $\bar{S}_{min}$ の節点 $\bar{v}$ に対しても, 同様に, $\bar{S}_{min}(\bar{v})$ を定義する. 最簡形ということより, 次の補題 2-1 が示される

[補題 2-1] S_{min} には $S_{min}(v_1) \equiv S_{min}(v_2)$ であるような異なる節点 v_1, v_2 は存在しない. $\bar{S}_{min}$ についても同様である.

(証明) S_{min} において v_2 に入っている枝を v_1 に入れ, 節点 v_2 を取り除いたプログラム形を考えると, そのプログラム形は S_{min} と強等価であり, S_{min} より節点数が一つ少なくなる. これは S_{min} が最簡形であることに矛盾する. (証明終り)

$\bar{S}_{min}$ に解釈 J , 初期値 ξ を与えたときの計算の道 (通過する節点名の系列) を $C(\bar{S}_{min}, J, \xi)$ と書き, その k 番目 (開始節点を 0 番とする) の節点を $C_k(\bar{S}_{min}, J, \xi)$ で表す. $C(S_{min}, J, \xi)$ も同様に定義し, そこから分岐節点を無視して (すなわち, 開始節点, 演算節点, 終了節点のみを取り出して), k 番目の節点を $C_k(S_{min}, J, \xi)$ で表す.

各 $\bar{v} \in \bar{V}$ に対し, $\phi(\bar{v})$ を

$C_k(\bar{S}_{min}, J, \xi) = \bar{v}$ となるようなすべての J, ξ, k について, 節点 $C_k(S_{min}, J, \xi)$ の集合と定義する. $\phi(\bar{v})$ は V の部分集合である.

[補題 2-2] $\bar{v} \in \bar{V}, v \in V$ に対し, $v \in \phi(\bar{v})$ となると, $\bar{S}_{min}(\bar{v}) \equiv S_{min}(v)$ である.

(略証) もし, ある解釈 J , 初期値 ξ 及び $k \geq 0$ について $C_k(\bar{S}_{min}, J, \xi) = \bar{v}, C_k(S_{min}, J, \xi) = v$ とし,

更に $\bar{S}_{min}(\bar{v}) \neq S_{min}(v)$ を示すような解釈 $\mathcal{J}'$, 初期値 ξ' が存在したと仮定すると, $\bar{S}_{min} \neq S_{min}$ を示すようなあるヘルブランドの解釈 $\mathcal{J}^H$, 初期値 ξ^H が以下に示すように作れるという議論により示せる。

ヘルブランドの解釈 $\mathcal{J}^H$ を作るため, まず, S_{min} のヘルブランド領域 $H_{S_{min}}$ をつぎのように定義する。もし x_i が S_{min} に現れるデータ変数, a_i が S に現れる个体定数なら “ x_i ” と “ a_i ” は $H_{S_{min}}$ の元である。 f_i が S_{min} に現れる関数記号で “ x_j ” が $H_{S_{min}}$ の元なら, “ $f_i(x_j)$ ” は H_S の元である。

ヘルブランドの解釈 $\mathcal{J}^H$ をつぎのように作る。 S_{min} に現れる各个体定数 a_i には $H_{S_{min}}$ の記号列 “ a_i ” を割当てる。 S_{min} に現れる各関数記号 f_i には $H_{S_{min}}$ の記号列 “ x_j ” を $H_{S_{min}}$ の記号列 “ $f_i(x_j)$ ” に写像するような関数を割当てる。

S_{min} の述語記号と个体定数 ξ', ξ'' への割当に関しては, $\bar{S}_{min}, S_{min}$ における $\mathcal{J}^H, \xi^H$ に対する計算の道が, そのはじめの k 番目の演算節点に至るまでは, それぞれ, $C(\bar{S}_{min}, \mathcal{J}, \xi), C(S_{min}, \mathcal{J}, \xi)$ に一致し, それ以降はそれぞれ, $C(\bar{S}_{min}(\bar{v}), \mathcal{J}', \xi'), C(S_{min}(v), \mathcal{J}', \xi')$ に一致するように行なう。(一般には, ヘルブランドの解釈には, 述語記号への割当に関してはなんの制限もないことに注意する) (略証終)

上の補題 2-1, 補題 2-2 を用いて以下に示す補題 2-3, 補題 2-4 がなりたつ。

[補題 2-3] $\alpha(\bar{v})$ は 1 個かつただ 1 個の節点を含む。

(証明) もし $v_1, v_2 \in \alpha(\bar{v})$ と仮定すると補題 2-2 より $\bar{S}_{min}(\bar{v}) \equiv S_{min}(v_1) \equiv S_{min}(v_2)$ がなりたつ。これは補題 2-1 より S_{min} が最簡形であることに矛盾する。(証明終)

[補題 2-4] 異なる $\bar{v}_1, \bar{v}_2 \in \bar{V}$ に対し, $\alpha(\bar{v}_1), \alpha(\bar{v}_2)$ が同じ $v \in V$ を含むことはない。

(証明) もし, $v \in \alpha(\bar{v}_1), v \in \alpha(\bar{v}_2)$ と仮定すると補題 2-2 より $S_{\min}(v) \equiv \bar{S}_{\min}(\bar{v}_1) \equiv \bar{S}_{\min}(\bar{v}_2)$ となりたつ。これは補題 2-1 より $\bar{S}_{\min}$ が最簡形であることに矛盾する。

補題 2-3, 補題 2-4 により, α は $\bar{V}$ から V への 1 対 1 対応と考えられる。定理の ρ をこの α とすると, 定理の (a), (b), (c) の成り立つことが以下のように示される。

(i) $S_{\min}$ と $\bar{S}_{\min}$ の開始節点, 終了節点, ループ節点をそれぞれ, v_E, v_I, v_L と $\bar{v}_E, \bar{v}_I, \bar{v}_L$ で表わす。 $v_E \in \alpha(\bar{v}_E), v_I \in \alpha(\bar{v}_I)$ であることは $S_{\min} \equiv \bar{S}_{\min}$ であることからあきらか。 $v_L \in \alpha(\bar{v}_L)$ であることは, $S_{\min}, \bar{S}_{\min}$ がともに最簡形であることからあきらか。ゆえに α は条件 (a) を満足する。

(ii) 補題 2-2 より α は条件 (b) を満足する。

(iii) 定理 (c) の L, L' について, $L \neq L'$ とすると $v_2' \neq v_2$ で $v_2' \in \alpha(\bar{v}_2)$ となる v_2' が V に存在し補題 2-3 に矛盾する。また, もし, $L \neq L'$ と仮定すると $\bar{v}_2' \neq \bar{v}_2$ で, $v_2 \in \alpha(\bar{v}_2')$ となる $\bar{v}_2'$ が $\bar{V}$ に存在することになり, これは補題 2-4 に矛盾する。ゆえに, $L = L'$ となり α は条件 (c) を満足する。

§ 2.5 結言

ヤ-ノフのプログラム形が与えられたとき、それと強
等価であり、演算節点と分岐節点の節点数の総和が最小
であるヤ-ノフのプログラム形を求める問題について考
察し、この問題がいわゆる“決定表”の問題を含んでい
ることを示した。したがって、この問題に対する能率の
よい解法はないと考えられる。他の等価性（たとえば、
演算および述語判定の実行順序の入れ換えを許した等価
性）の定義の下での最簡形を求める能率のよい手続きを
考えることが今後にのこされた問題である。

第3章 サブルーチン化によるプログラム形の簡単化

§3.1 序言

システムプログラム等においてはプログラムの長さを短くすることは重要なことであり、また、理論的にもプログラム長の短縮問題は興味深い。

この章では、有向グラフで表現されたヤーノフのプログラム形^(*)を、複数の入口点を持ちうるサブルーチンを用いて書直す問題を考える。多入ロサブルーチン化は有効なプログラム短縮技法であり、実際にもよく使われている。計算実行の順序を制御するためにプッシュダウンスタック（または単にスタック）を全体として一つ設け、各サブルーチンへ入るときスタック命令により戻り先アドレスをスタックしておき、サブルーチンの終りのリターン命令でそのときのスタックのトップのアドレスへ戻る。もちろん、サブルーチンは入れ子構造になりうるが、どのがヤーノフタイプ（有限オートマトンの的）であるため、再帰的（リカーシブ）なコールは使用しないですむ。

開始節点から終了節点に至るまでの途中の演算と述語判定の結果の記号系列が集合として等しいとき（スタック命令やリターン命令の実行は無視する）、二つのプログラム形は動作等価であるということにする。演算や述語判定の実行系列を保存してプログラムを書直すということとは、たとえば、システムプログラムをアセンブリ言語で書いて実行順序としてはすでになんらかの意味で最適化されている、あるいは、実行順序を変更できないといった場合に、さらに、共通部を見つけてまとめるということに相当している。

この章では、任意のヤーノフのプログラム形が与えら

れたとき、そのと動作等価で、演算節点と分岐節点の個数の和が最小の多入力サフルーチンを許した（すなわち、ストップ命令、リターン命令を使用した）プログラム形を求める問題を考察し、その最簡形が $|V|^2 \log |V|$ ($|V|$ は与えられたプログラム形の節点数) に比例する手数で構成できる手続きを示した。

§ 3.2 定義と問題

(一変数の)関数記号の集合を F , 述語記号の集合を P とする。(F, P 上の) ヤーノフのプログラム形 S とは, その命令が図3-1に示されるような形をしたプログラム形をいう⁽¹⁾。ただし, 開始節点はただ一つである。演算節点はある関数記号 $f_i \in F$ をラベルとしてもち, 分岐節点から出る二本の枝はある述語記号 $P_j \in P$ およびその否定形 $\bar{P}_j$ をラベルとしてもち。(2章で定義したヤーノフのプログラム形では, ループ節点が許されてゐる。)

このプログラム形 S に現れてゐる関数記号および述語記号にいわゆる“解釈”を与え, 開始節点において内部変数にある初期値を代入すると, “計算”が実行される⁽²⁾。演算節点では, ラベルとして書かれた関数記号が示す演算を実行する。ラベル P_j および $\bar{P}_j$ の二本の出力枝をとつ分岐節点では, P_j による述語判定を行ない, 真ならラベル P_j の枝を選び, 偽ならラベル $\bar{P}_j$ の枝を選び, 計算の制御をつぎの節点に移す。そして, 終了節点では変数の内容を出力して停止する。

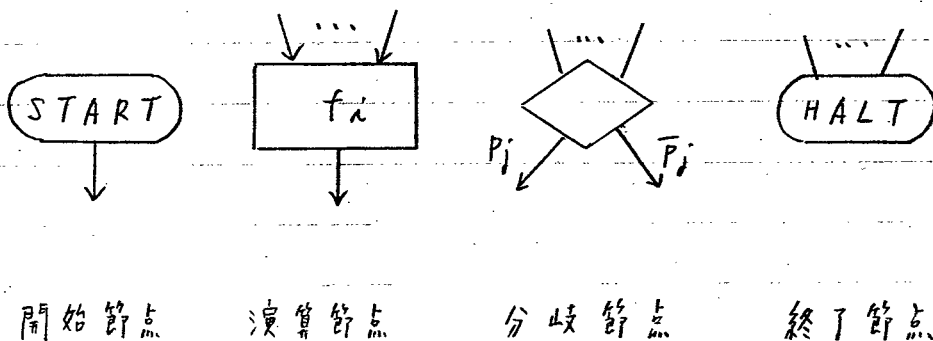


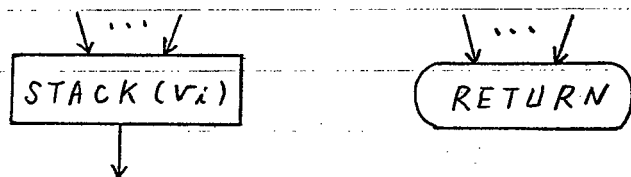
図 3-1 ヤーノフのプログラム形の命令

簡単のため、つぎの仮定をおく。

[A1: 各節点および枝について、開始節点からその節点またはその枝を通過して終了節点に至るような計算の道（ある解経、初期値を与えたときの道）が存在する。]

A1は、各節点について、開始節点からその節点またはその枝を通過して終了節点に至るようなグラフ上の道（枝の向きに沿った節点と枝の系列）が存在し、そのような道の道においても、演算することなく同じ述語判定を2度以上しないということである。ヤーノフのプログラム形は常にこのように直せる。⁽⁷⁾ このヤーノフのプログラム形のクラスを $\mathcal{A}$ とする。

つぎに、ヤーノフのプログラム形に、多入力のサブルーチンを許したプログラム形を定義する。このプログラム形は、計算実行の順序を制御する機能として、プッシュダウンスタックを全体として一つ設け、いわゆるスタック、リターン命令を使用する。このプログラム形は図3-1の開始節点、終了節点、演算節点、分岐節点以外に、命令の形が図3-2に示されているようなスタック節点とリターン節点からなる有向グラフ（必ずしも、連結でない）である。



スタック節点

リターン節点

図3-2 スタック節点とリターン節点

計算の制御の流れは、命令 $STACK(v_i)$ をもつスタック節点では、そこで指定された節点名 v_i をプッシュダウンスタックのトップにスタックしてからつぎの節点に制御を移す。リターン節点では、スタックのトップに書かれている節点名の節点に制御を移し、スタックをポップアップして、トップの節点名を一つ取り除く。開始節点、演算節点、分岐節点、終了節点での動作は、ヤードのプログラム形と同じである。開始節点では、プッシュダウンスタックは空とする。このサブルーチンを許したプログラム形のクラスを R とする。もちろん、 $\mathcal{A} \subseteq R$ と考える。

R 内のプログラム形について、便宜上、つぎの仮定をおく。

[A2: 各節点および枝について、開始節点からその節点またはその枝を通過して終了節点に至るような計算の道が存在する。]

[A3: 終了節点に制御が移ったときは、スタックは空である。また、スタックが空のときには、リターン節点に制御は移らない。]

プログラム形 $S \in \mathcal{A}$ のある計算（ある解釈と初期値を与えて）の実行において、節点 v_1 から v_2 に至る間に実行した関数記号 f_i と述語の判定結果 P_j および $\bar{P}_j$ の記号系列（ v_1 が開始節点のときは“START”という記号も含める）が w であるとき（ただし、途中で節点 v_2 を通ったときのラベルは含めるが最後に節点 v_2 に至ったときの v_2 のラベル（演算節点のときの関数記号）は含めない）、 $v_1 \xrightarrow{w} v_2$ と書く。また、 $L(v_1, v_2) = \{w \mid v_1 \xrightarrow{w} v_2, \text{かつ、途中で節点 } v_2 \text{ は通らない}\}$ とする。

プログラム形 $S \in R$ の任意の節点 v に対して、 v

から、 v に制御があったときのスタックのトップが $RETURN$ 命令によりはじめてポップアップされる時点までの計算の道 (v においてスタックが空のときは終了節点に至るまでの計算の道) における, “START” および関数記号と述語の判定結果の記号系列 (以下, 記号系列というときは, このように “STACK (v_i)” と “RETURN” は除外している) w の集合を $T(v)$ とする。

任意の解釈, 初期値について, プログラム形 S, S' の対応する計算の道において, 途中で実行した演算および判定結果が記号系列として等しいとき, S と S' は動作等価という (以下, 略して等価という.) 仮定 A1, A2 より, これは, 開始節点から終了節点に至るグラフ上の (演算と述語の判定結果の) 記号系列の集合が等しいことと一致する。

さて, プログラム形の簡単さの尺度として “コスト” を導入する。演算節点および分岐節点のコストを C_1 , スタック節点のコストを C_2 , リターン節点, 開始節点, 終了節点に対するコストを, それぞれ, C_3, C_4, C_5 とする。これらはそこで実行する命令を, たとえば, 機械語系列で表現したときのルーチンの長さに対応していると考えられる。プログラム形のコストを, 上記の節点のコストにそれぞれの個数を乗じたものの和と定義する。

図 3-3, 図 3-4 に, それぞれ, クラス $\mathcal{R}$ に属するプログラム形の例 S_1, S_2 を示す。 S_1 と S_2 は等価であり, S_1 のコストは $11C_1 + C_4 + C_5$, S_2 のコストは, $7C_1 + 7C_2 + 3C_3 + C_4 + C_5$ である。

この章では, 任意のプログラム形 $S \in \mathcal{R}$ が与えられたとき, S と等価な, $\mathcal{R}$ 内のプログラム形でコスト最小のものを一つ求める問題を考察する。しかし, 一般に

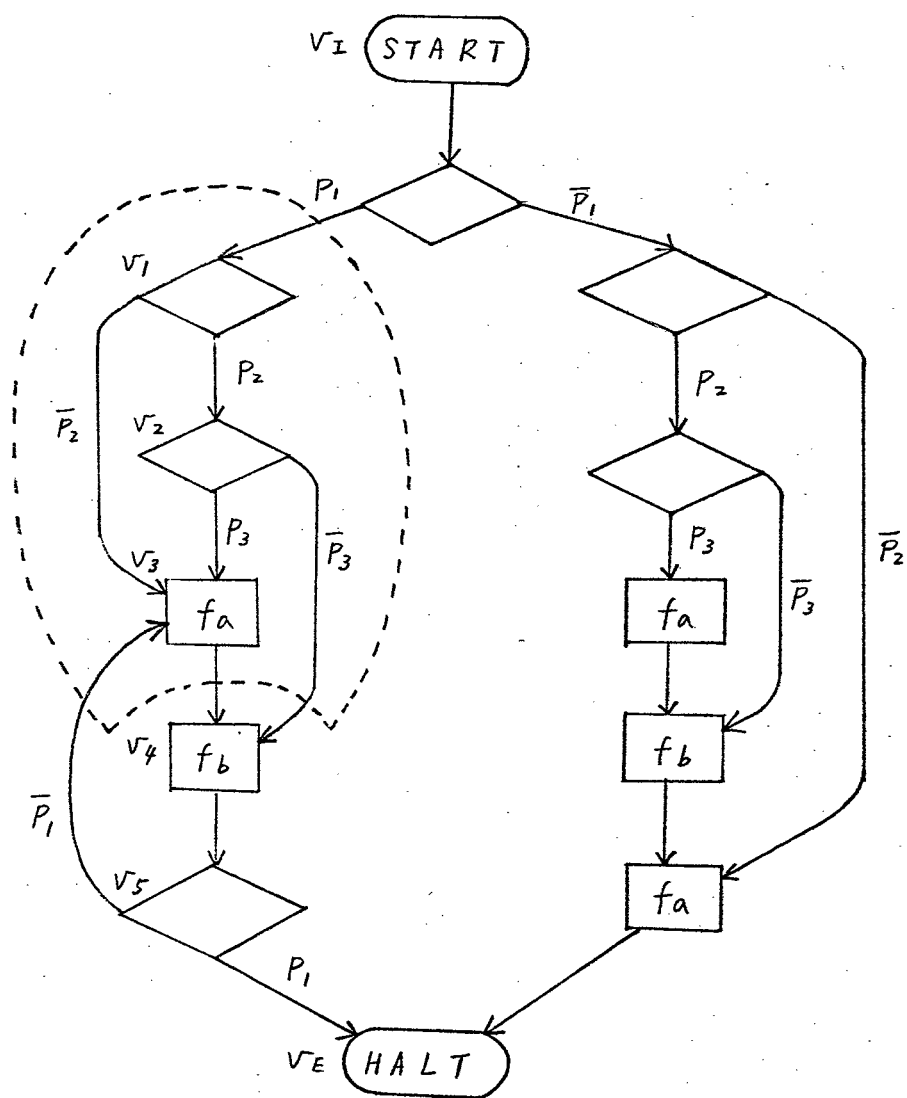


図 3-3 $\mathcal{H}$ に属するプログラム形の例 S_1

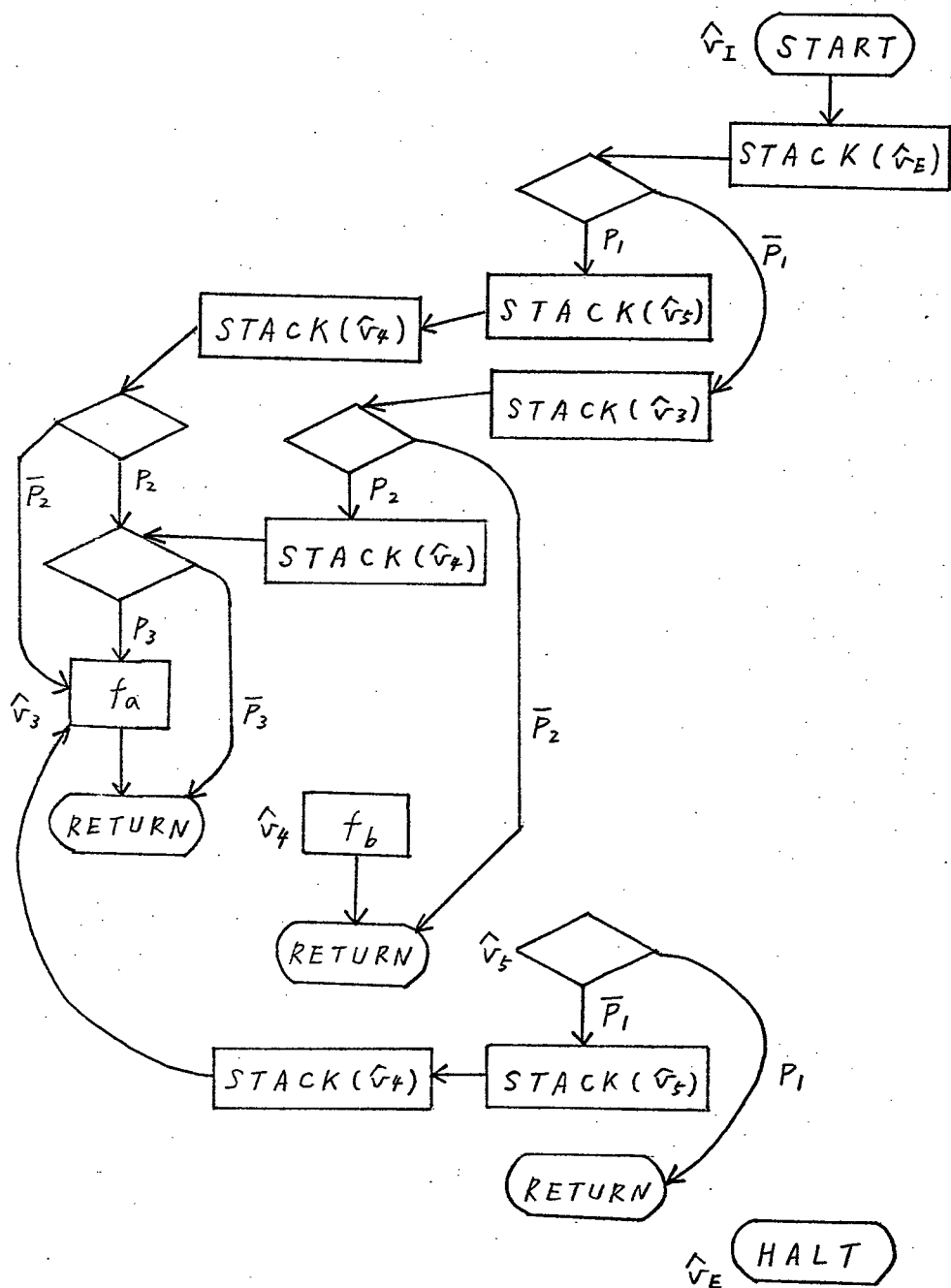


図 3-4 R に属するプログラム形の例 S_2

は、能率のよいアルゴリズムはないので、以下では、お
よそに、 $C_2 = 0$ とし、すなわち、STACK 節点のコス
トを無視して、コスト最小のもの⁺を求める方法につい
て議論する。

⁺いいかえれば、演算節点と分岐節点の個数の和が最小
のものである。(リターン節点、終了節点は1個ずつでよ
い。) 演算節点や分岐節点をマクロ的な命令とみなせば、
これらの個数がプログラム長に影響すると考えられる。

クラス Π のプログラム形に関して、有向グラフとしていくつかの概念を定義する。節点 v に対し、 v から終了節点に至るすべての道が共通に(かならず)通る節点を v のサブエンドという(これらの節点は、枝の向きを逆にし、 v との終了節点を開始節点とみたとき、文献(10)における v の *dominator* となっている)。 v を除いて最初に通るサブエンドを $E^1(v)$ ($E(v)$ と書く。), 一般に、 k 番目に通るサブエンドを $E^k(v)$ と書く。サブエンドの定義からあきらかなように、 v の k_1 番目のサブエンド $E^{k_1}(v)$ の k_2 番目のサブエンド $E^{k_2}(E^{k_1}(v))$ と、 v の $k_1 + k_2$ 番目のサブエンド $E^{k_1+k_2}(v)$ は同じ節点になる。節点 v に対して、 $E^k(v)$ が終了節点のとき、 k を v のレベルといい、 $\ell(v)$ と書く(ただし、例外として、開始節点、終了節点のレベルは 0 とする)。たとえば、図 3-3 において、 v_1 のサブエンドは $E^1(v_1) = v_4$, $E^2(v_1) = v_5$, $E^3(v_1) = v_E$ である。

節点 v に対し、 v をサブエンドとするような節点の集合 $\{v' \mid E^k(v') = v, k=1, 2, \dots\}$ が空でないとき、これらの節点と節点間の枝で構成される部分グラフのことを、 v を終点とするサブルーチン可能グラフという。

ただし、これらの節点から外部へ出る枝(定義より、節点 v へ入る枝のみである)も含める。たとえば、図 3-3 において、 v_4 を終点とするサブルーチン可能グラフは破線で囲んだ部分である。定義より、つぎの補題 3-1 がなりたつ。

[補題 3-1] サブルーチン可能グラフは入小子構造になっている。すなわち、異なる二つのサブルーチン可能グラフは、互いに共通な節点を持たないか、あるいは、一方が他方を完全に含む。

§ 3.4 サブルーチン化の条件

与えられたプログラム形 $S \in \mathcal{F}$ と等価な、クラス $\mathcal{F}$ における最簡形 $\bar{S}$ について、最簡形ということより、つぎの補題3-2, 3-3 がなりたつ。 $\bar{S}$ の開始節点, 終了節点をそれぞれ, $\bar{v}_I, \bar{v}_E$ とする。

[補題 3-2] $\bar{S}$ の異なる 2 つの節点 $\bar{v}_1, \bar{v}_2$ に対して, $L(\bar{v}_1, \bar{v}_E) = L(\bar{v}_2, \bar{v}_E)$ となることはない。

[補題 3-3] $\bar{S}$ において, 2 つの節点 $\bar{v}_1, \bar{v}_2$ とある $k_1, k_2 > 0$ に対し, $L(\bar{v}_1, E^{k_1}(\bar{v}_1)) = L(\bar{v}_2, E^{k_2}(\bar{v}_2))$ ならば, $k_1 = k_2$ であり, さらに, $L(\bar{v}_1, E(\bar{v}_1)) = L(\bar{v}_2, E(\bar{v}_2)), L(E(\bar{v}_1), E^2(\bar{v}_1)) = L(E(\bar{v}_2), E^2(\bar{v}_2)), \dots, L(E^{k_1-1}(\bar{v}_1), E^{k_1}(\bar{v}_1)) = L(E^{k_2-1}(\bar{v}_2), E^{k_2}(\bar{v}_2))$ でなければならない。

(補題3-3の略証) とし, $\omega', \omega'' \in L(\bar{v}_1, E(\bar{v}_1)), \bar{v}_2 \xrightarrow{\omega'} \bar{v}_2', \bar{v}_2 \xrightarrow{\omega''} \bar{v}_2'', \bar{v}_2' \neq \bar{v}_2''$ なる ω', ω'' が存在したと仮定すると, 条件 $L(\bar{v}_1, E^{k_1}(\bar{v}_1)) = L(\bar{v}_2, E^{k_2}(\bar{v}_2))$ を用いて, $L(\bar{v}_2', E^{k_2}(\bar{v}_2)) = L(\bar{v}_2'', E^{k_2}(\bar{v}_2))$, したがって, $L(\bar{v}_2', \bar{v}_E) = L(\bar{v}_2'', \bar{v}_E)$ となり, 補題3-2 に反する。また, 上と対称な議論を使うことにより, $L(\bar{v}_1, E(\bar{v}_1)) = L(\bar{v}_2, E(\bar{v}_2))$ が示される。これを繰返せば, 補題3-3 が証明される (略証終)

クラス $\mathcal{R}$ のプログラム形 S^+ (開始節点, 終了節点をそれぞれ v_I, v_E とする) のスタック節点, リターン節点以外の各節点 v_i に対し, $\bar{S}$ の節点の集合 $\mathcal{V}$ の $(\bar{v}_i)$ をつぎのように対応させる。

+ S^+ において, 遷移が遷移前のスタックの内容に関係しないとき, “ $\rightarrow$ ” なる記号を用いる。

ある記号系列 w について, $\mathcal{G}$ において, $\bar{v}_i \xrightarrow{w} \bar{v}$, $\bar{S}$ において, $\bar{v}_i \xrightarrow{w} \bar{v}$ であるなら, $\bar{v} \in \mathcal{O}(\bar{v}_i)$ である. 逆に, $\bar{v} \in \mathcal{O}(\bar{v}_i)$ なら, そのような w が存在する.

つぎの補題 3-4, 補題 3-5 は, クラス $\mathcal{R}$ における最簡形 $\bar{S}$ とクラス $\mathcal{R}$ の任意の等価なプログラム形 $\mathcal{G}$ との関係を表述している.

[補題 3-4] $\mathcal{G}$ の節点 $\bar{v}$, $\bar{S}$ の節点 $\bar{v}_i$ に対し, $\bar{v}_i \in \mathcal{O}(\bar{v})$ とすると, ある $k_i > 0$ が存在して, $T(\bar{v}) = L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ がなりたつ.

証明は付録に示す.

[補題 3-5] $\bar{S}$ の節点 $\bar{v}_i, \bar{v}_j$ について, $\bar{v}_i, \bar{v}_j \in \mathcal{O}(\bar{v})$ となるような $\mathcal{G}$ の節点 $\bar{v}$ が存在すれば, $L(\bar{v}_i, E(\bar{v}_i)) = L(\bar{v}_j, E(\bar{v}_j))$ でなければならぬ.

(証明) $\bar{v}_i, \bar{v}_j \in \mathcal{O}(\bar{v})$ と補題 3-4 より, ある $k_i, k_j > 0$ について, $L(\bar{v}_i, E^{k_i}(\bar{v}_i)) = L(\bar{v}_j, E^{k_j}(\bar{v}_j))$ となる. ゆえに, 補題 3-3 より, $L(\bar{v}_i, E(\bar{v}_i)) = L(\bar{v}_j, E(\bar{v}_j))$ でなければならぬ. (証明終)

さて, $\bar{S}$ の節点の分割 $\pi(\bar{S})^+$ をつぎのように定義する. $L(\bar{v}_i, E(\bar{v}_i)) = L(\bar{v}_j, E(\bar{v}_j))$ のとき, おつきのときのみ, $\bar{v}_i$ と $\bar{v}_j$ は同じブロック (クラス) に属する (開始節点と終了節点は, それぞれ, それ一つでブロックをなす).

3.5 節では, 分割 $\pi(\bar{S})$ の各ブロックを一つの節点に対応させ, $\bar{S}$ と等価なクラス $\mathcal{R}$ 内のプログラム形 $\tilde{S}$ を構成するアルゴリズムを示す. そして, $\tilde{S}$ は $C_2 = 0$ と考えたときコストが最小になっていることを示す.

$^+ \bar{S}$ の節点の集合を $\bar{V}$ とし, $\pi(\bar{S}) = \{C_1, \dots, C_m\}$, 各 $C_i \subseteq \bar{V}$ とおくと, $C_1 \cup \dots \cup C_m = \bar{V}$, $i \neq j$ ならば $C_i \cap C_j = \emptyset$ である.

§ 3.5 R 内の最簡形の構成アルゴリズム

ここでは、 $C_2 = 0$ と考え、任意の $S \in \mathcal{R}$ が与えられたとき、それと等価でコスト最小の R に属するプログラム形 $\bar{S}$ を構成する方法を述べる。このアルゴリズムは、 S の節点数を $|V|$ とすると、 $|V|^2 \log |V|$ に比例する手数で終了する。アルゴリズムは4つの手続きに分かれている。

3.5.1 S から $\bar{S}$ への変換

与えられた S を、それと等価な、 $\mathcal{R}$ における最簡形 $\bar{S}$ へ変換する。それは、 S を“決定性有限オートマトン”とみて、その最簡形を求める問題と同じであり、後者については、すでに、手数が $|V| \log |V|$ に比例する手続きが知られている⁽⁸⁾ のでそれを利用する。

[手続き 3-1] S を有限オートマトンとみて、文献(8)の方法で、最簡形 $\bar{S}$ を求める。

あきらかに、 $\bar{S}$ の節点数は n 以下である。

3.5.2 $\bar{S}$ の各節点 $\bar{x}$ のサブエンドを求める。

ここでは、 $\bar{S}$ の各節点 $\bar{x}$ に対し、 $\bar{x}$ のサブエンド $E(\bar{x})$ およびレベル $l(\bar{x})$ を求める。

サブエンドの定義は、 $\bar{S}$ の枝を逆にすれば文献(10)における dominator の定義と一致するので、文献(10)のアルゴリズムによりサブエンドが求められる。

[手続き 3-2] $\bar{S}$ の枝を逆にし、 $\bar{S}$ の枝を逆にし、 $\bar{S}$ の終了節点を開始節点とみなして、文献(10)のアルゴリズムを適用して、各節点 $\bar{x}$ に対してサブエンド $E(\bar{x})$ を

求める。また、各節点に対するレベルを、この結果を利用することによって、 $\bar{S}$ の終了節点から逆にいわゆる depth-first-search の方法⁽¹⁰⁾ でたどり求める。

[補題 3-6] 上記の手続き 2 によって、すべての節点 $\bar{v}$ に対し、サブエンド $E(\bar{v})$ とレベル $l(\bar{v})$ が手数 $O(|V| \log |V|)$ で求められる。

(略証) 文献 (10) のアルゴリズムは手数 $O(|V| \log |V|)$ であり、すべての節点に対して、そのレベルを求める手数は $O(|V|)^+$ である。

3.5.3 $\bar{S}$ の節点の分割 $\pi(\bar{S})$ を求める。

手続き 3-3 では、 $L(\bar{v}_i, E(\bar{v}_i)) = L(\bar{v}_j, E(\bar{v}_j))$ となる $\bar{v}_i, \bar{v}_j$ が同一のブロックに入るような $\bar{S}$ の節点の分割 $\pi(\bar{S})$ を求める。

[手続き 3-3]

(ステップ 1) まず、 $\bar{S}$ の各節点 $\bar{v}$ (開始節点を除いた節点) に対して、節点 $[\bar{v}, 1], [\bar{v}, 2], \dots, [\bar{v}, l(\bar{v})]$ を設け、 $\bar{v}$ が演算節点のときには、 $\bar{v}$ に付いている関数記号をそれらの節点に付ける。さらに、 $\bar{v}$ があるサブルーチン可能グラフの終点になっているときには、節点 $[\bar{v}, l(\bar{v}) + 1]$ も設け、その節点にはラベルを付けない。

$\bar{S}$ において、節点 $\bar{v}$ から $\bar{v}'$ に枝があれば、それぞれの k について、節点 $[\bar{v}, k]$ から $[\bar{v}', k]$ に枝を結ぶ。もし、 $\bar{v}$ から $\bar{v}'$ への枝に述語の判定結果の記号 (ラベル) があれば、 $[\bar{v}, k]$ から $[\bar{v}', k]$ への枝にも同じラベルを付ける。それぞれの k について、節点 $[\bar{v}, k]$

⁺ $O(f(n))$ は $f(n)$ のオーダー ($f(n)$ に比例する) の意味。

に対して、節点 $[\bar{v}', k]$ が存在することは、 $\bar{v}'$ があるサブルーチン可能グラフの終点であるときでも、 $l(\bar{v}) = l(\bar{v}') + 1$ であることからあきらかである。

(ステップ ii) $[\bar{v}, l(\bar{v}) + 1]$ の形の節点だけからなる節点の集合を B_0 、残りのすべての節点の集合を B_1 とし、 B_0 を文献 (8) での最終状態の集合とみなして、文献 (8) の状態数最小化アルゴリズムを適用する。

(ステップ iii) ステップ ii の終了時、 $[\bar{v}_1, l(\bar{v}_1)]$ と $[\bar{v}_2, l(\bar{v}_2)]$ が同じブロックに含まれているならば、 $\bar{v}_1$ と $\bar{v}_2$ を同じブロックに入れて、 $\bar{v}$ の節点を分割する (開始節点、終了節点は、それぞれ、それぞれ一つでブロックをなす)。

[補題 3-7] 手続き 3-3 によって求められた分割は、 $\pi(\bar{v})$ である。また、手続き 3-3 の手数は、 $O(|V|^2 \log |V|)$ である。

(証明) ステップ i の結果、 $L(\bar{v}, E(\bar{v})) = L([\bar{v}, l(\bar{v})], [E(\bar{v}), l(\bar{v})])$ となっている。また、ステップ ii の結果 $L([\bar{v}_1, l(\bar{v}_1)], [E(\bar{v}_1), l(\bar{v}_1)]) = L([\bar{v}_2, l(\bar{v}_2)], [E(\bar{v}_2), l(\bar{v}_2)])$ のときまたそのときにかきつて、 $[\bar{v}_1, l(\bar{v}_1)]$ と $[\bar{v}_2, l(\bar{v}_2)]$ が同じブロックに入れられている。ゆえに、ステップ iii による分割は $\pi(\bar{v})$ である。

ステップ i の手数は $O(|V|^2)$ である。ステップ ii は、文献 (8) のアルゴリズムを利用しているが、この手数は節点数を N とすると、 $O(N \log N)$ である。しかし、ステップ ii で扱う節点数は $|V|^2$ 以下であるため、手数は $O(|V|^2 \log |V|)$ である。また、ステップ iii の手数は $O(|V|^2)$ である。
(証明終)

3.5.4 $\tilde{\Sigma}$ の定義

分割元 $(\bar{S})$ において, $\bar{S}$ の節点 $\bar{v}$ を含むブロックを $C(\bar{v})$ と書く.

[手続き 3-4] $\pi(\bar{S})$ の各ブロックに対応して, $\tilde{\Sigma}$ の節点 (スタック節点, リターン節点以外) を設ける (簡単のため, ブロックと節点を同一視する). $\bar{v}$ が $\bar{S}$ の演算節点のときには, $C(\bar{v})$ を演算節点とし, $\bar{v}$ と同じ関数記号を付ける. $\bar{v}$ が $\bar{S}$ の分岐節点のときには, $C(\bar{v})$ を分岐節点とする (ラベルは付けない). $C(\bar{v}_L)$, $C(\bar{v}_R)$ を, それぞれ, $\tilde{\Sigma}$ の開始節点, 終了節点とする. リターン節点を一つ設ける.

$\bar{S}$ において, 節点 $\bar{v}$ から $\bar{v}'$ へ枝があるとき, $C(\bar{v})$ から $C(\bar{v}')$ への遷移を以下のように定める.

(i) $l(\bar{v}) = l(\bar{v}')$ のとき, 節点 $C(\bar{v})$ から $C(\bar{v}')$ へ直接枝を結ぶ, $\bar{v}$ から $\bar{v}'$ への枝に述語の判定結果の記号 (ラベル) があれば, $C(\bar{v})$ から $C(\bar{v}')$ への枝に同じラベルを付ける.

(ii) $l(\bar{v}) < l(\bar{v}')$ のとき, $k = l(\bar{v}') - l(\bar{v})$ とおくと, $C(\bar{v})$ から $C(\bar{v}')$ へ至る途中に k 個のスタック節点 $STACK(C(E^k(\bar{v}')))$, $\dots$, $STACK(C(E'(\bar{v}')))$ を順に設ける ($E'(\bar{v}')$, $\dots$, $E^k(\bar{v}')$ は, 手続き 3-2 における結果を利用して求められる). $\bar{v}$ から $\bar{v}'$ への枝にラベルがあれば, $C(\bar{v})$ から $STACK(C(E^k(\bar{v}')))$ への枝に同じラベルを付ける.

(iii) $l(\bar{v}) > l(\bar{v}')$ のとき, 節点 $C(\bar{v})$ からリターン節点に枝を結ぶ, $\bar{v}$ から $\bar{v}'$ への枝にラベルがあれば, この枝にも同じラベルを付ける.

[補題 3-8] 上記の手続き 3-4 の $\tilde{\Sigma}$ の定義は矛盾なく行え, 手続き 3-4 の手数は, $O(|V|^2)$ 以下である.

(略証) (ii) の $C(E^k(\bar{v}'))$, ..., $C(E(\bar{v}'))$ がフロック $C(\bar{v})$ 内の他の節点に対してそれぞれ同一になることが, $\pi(\bar{s})$ の定義と補題 3-3 から証明できる。(ii) 以外に関しては, $\pi(\bar{s})$ の定義からあきらか。(略証終)

$\bar{S}$ において、つぎのような 3 つ組 $(\bar{v}, \bar{w}, \bar{v}')$ の集合を $\bar{M}_i$ とする。 $\bar{v} \xrightarrow{\bar{w}} \bar{v}'$ であり、遷移 $\bar{v} \xrightarrow{\bar{w}} \bar{v}'$ の途中の節点のレベルは $l(\bar{v})$ より小さくなく、 $\bar{v}'$ のレベル $l(\bar{v}')$ は $l(\bar{v})$ と等しい。そして、遷移の途中での節点のレベルの最大とレベル $l(\bar{v})$ とのレベル差が ϵ である。

また、 $\tilde{S}$ において、つぎのような 3 つ組 $(\tilde{v}, \tilde{w}, \tilde{v}')$ の集合を $\tilde{M}_i$ とする (ただし、 $\tilde{v}, \tilde{v}'$ は、スタック節点、リターン節点以外の節点である)。 $\tilde{v}$ から記号系列 $\tilde{w}$ が付いた遷移の行き先が $\tilde{v}'$ であり、遷移の途中では、 $\tilde{v}$ におけるスタックのトップがポップアップされることなく、遷移後の節点 $\tilde{v}'$ におけるスタックの内容は、遷移前の $\tilde{v}$ におけるスタックの内容と同じである。そして、この遷移の途中でスタックが最も深くなる所では $\tilde{v}$ におけるスタックの深さよりさらに ϵ だけ深くなっている。

$\bar{M}_i$ と $\tilde{M}_i$ との間に、補題 3-9 がなりたつ。

[補題 3-9] $(\bar{v}, \bar{w}, \bar{v}') \in \bar{M}_i$ であるば、 $(C(\bar{v}), \bar{w}, C(\bar{v}')) \in \tilde{M}_i$ がなりたち、また逆に、 $(\tilde{v}, \tilde{w}, \tilde{v}') \in \tilde{M}_i$ であるば、 $\tilde{v} = C(\bar{v})$ である任意の $\bar{v}$ に対して、 $\tilde{v}' = C(\bar{v}')$ であるような $\bar{v}'$ が存在して、 $\bar{v}$ と $\bar{v}'$ の間に $(\bar{v}, \bar{w}, \bar{v}') \in \bar{M}_i$ の関係がなりたつ。

この補題 3-9 は ϵ についての帰納法により証明できるが、詳細は省く。

[補題 3-10] $\tilde{S}$ は $\bar{S}$ と等価である。

(略証) $\tilde{S}$ の構成のしかたと補題 1 より、 $\tilde{S}$ が条件 A3 を満足することが証明できる。ゆえに、 $\tilde{S}$ における開始節点 $\tilde{v}_E$ から終了節点 $\tilde{v}_E$ に至る道に付けられた記号系列 $\tilde{w}$ は、ある $\bar{v}$ に対して、 $(\tilde{v}_E, \tilde{w}, \tilde{v}_E) \in \tilde{M}_i$

かなりたつ。 $\tilde{v}_E = C(\bar{v}_E)$, $\tilde{v}_E = C(\bar{v}_E)$ であることから、補題 3-9 より、 $\tilde{S}$ は $\bar{S}$ と等価となる。(略証終)

S と等価なクラス R の任意のプログラム形 $\tilde{S}$ より $\tilde{S}$ の方が (スタック, リターン節点以外の) 節点数が多くないことを以下で示す。

$\tilde{S}$ の各節点 $\tilde{v}$ (スタック, リターン節点以外の) に対応する $\bar{S}$ の節点の集合 $\phi(\tilde{v})$ と $\tilde{S}$ の各節点 (スタック, リターン節点以外の) を表わしている $\pi(\bar{S})$ のブロックとの関係を補題 3-11 で述べる。

[補題 3-11] $\tilde{S}$ のスタック, リターン以外の節点 $\tilde{v}$ に対して, $\phi(\tilde{v}) = \{\bar{v}_1, \bar{v}_2, \dots, \bar{v}_h\}$ とすると, $\bar{v}_1, \bar{v}_2, \dots, \bar{v}_h$ は, すべて, 分割 $\pi(\bar{S})$ のある一つのブロックに含まれる。

補題 3-11 の証明は, 補題 3-5 と分割 $\pi(\bar{S})$ の定義より, あきらか。

[定理 3-1] $C_2 = 0$ の場合, $\tilde{S}$ は, 与えられた $S \in R$ と等価で, クラス R で, 演算節点と分岐節点の個数の和が最小のものである。

(証明) S と等価なクラス R の任意のプログラム形 $\tilde{S}$ を考える。 $\tilde{S}$ と $\bar{S}$ が等価であり, $\bar{S}$ が条件 A1 を満たすことより, $\bar{S}$ の任意の節点 $\bar{v}$ に対して, $\bar{v} \in \phi(\tilde{v})$ となるような $\tilde{S}$ の節点 $\tilde{v}$ が存在する。これと補題 3-11 より, $\tilde{S}$ の節点数は $\pi(\bar{S})$ のブロック数より少なくない。(証明終)

$C_2 = 0$ の場合の任意の最簡形について, その最簡形の各節点 $\tilde{v}$ に対して $\phi(\tilde{v})$ を考えると, 補題 3-11, 定理 3-1 より, $\phi(\tilde{v})$ の全体は分割 $\pi(\bar{S})$ に一致する。

§ 3.7 結言

任意のチャーノフのプログラム形が与えられたとき、それと動作等価で、演算節点と分岐節点の個数の和が最小（スタック節点やリターン節点を無視し）の多入ロサフルーチンを許したプログラム形を求める問題を考察し、与えられたチャーノフのプログラム形の節点数を $|V|$ とすると、その最簡形が $|V|^2 \log |V|$ に比例する手数で構成できる手続を示した。

スタック節点とリターン節点を含めた節点数の総和を最小化する問題においては、一カ所からしか呼ばれないサフルーチンはサフルーチン化しなかつたり、スタック動作を行なう位置を変更したり、スタック節点のみからなるサフルーチンを作ったりしてスタック節点の総和をへらせる場合もあるが、最適なもの（節点数の総和が最小のもの）を求める能率のよい方法は見つかっていない。これは今後にのこされた問題である。

§ 3.8 付録 補題 3-4 の証明

まず, $\bar{v}_i \in \alpha(\hat{G})$ であることより, ある w_i に対して, $\hat{G}_1 \xrightarrow{w_i} \hat{G}$, $\bar{v}_1 \xrightarrow{w_i} \bar{v}_i$ がなりたつ (このことは以下で断わりなく使用する)

$T(\hat{G}) \subset L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ であることを, 以下の (i), (ii), (iii), (iv) を述べて証明する.

(i) $\hat{G}$ は条件 A2 を満足しており “無駄” な節点を含まない。また, $\bar{S}$ と $\hat{G}$ は等価である。ゆえに, $\hat{G}$ における任意の $w \in T(\hat{G})$ に対して, $\bar{S}$ においては, 節点 $\bar{v}_i \in \alpha(\hat{G})$ から, 系列 w が付いた遷移 $\bar{v}_i \xrightarrow{w} \bar{v}_i'$ が存在する。

(ii) 節点 $\bar{v}_i \in \alpha(\hat{G})$ から, 系列 $w \in T(\hat{G})$ が付いた遷移の行先の節点 $\bar{v}_i'$ ($\bar{v}_i \xrightarrow{w} \bar{v}_i'$) は w に依存せず同一の節点である。

(ii の証明) $w', w'' \in T(\hat{G})$ とすると, (i) より, 遷移 $\bar{v}_i \xrightarrow{w'} \bar{v}_i'$, $\bar{v}_i \xrightarrow{w''} \bar{v}_i''$ が存在し, 以下で示すように, $L(\bar{v}_i', \bar{v}_E) = L(\bar{v}_i'', \bar{v}_E)$ となる (図 3-5 参照)。

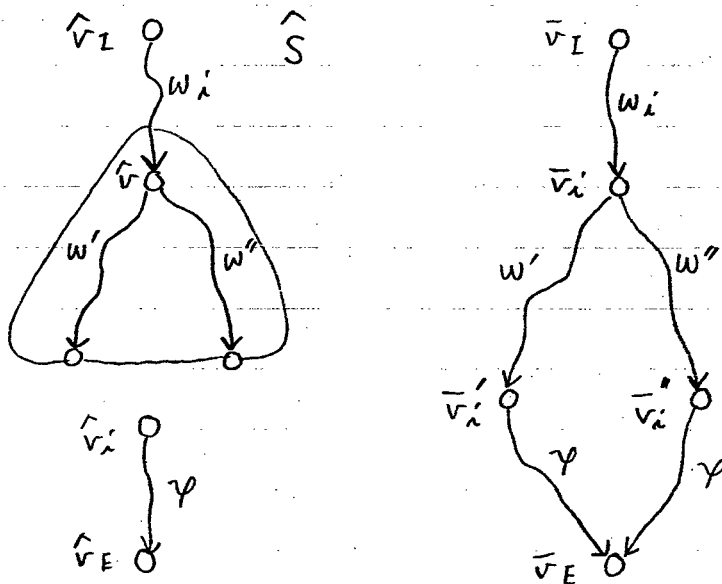


図 3-5 補題 3-4-ii の証明の説明図

ゆえに、補題 3-2 より $\bar{v}_1', \bar{v}_1''$ は同一節点でなければならぬ。

$L(\bar{v}_1, \bar{v}_E)$ の任意の系列 γ に対して、 $w_1, w', \gamma \in L(\bar{v}_1, \bar{v}_E)$ がなりたつ。今と $\bar{S}$ は等価、すなわち、 $T(\hat{v}_1) = L(\bar{v}_1, \bar{v}_E)$ より、 $w_1, w', \gamma \in T(\hat{v}_1)$ がなりたつ。遷移 $\hat{v}_1 \xrightarrow{w_1} \hat{v}_1'$ のあと、 $\hat{v}_1'$ から系列 $w' \in T(\hat{v}_1)$ による遷移の行き先を $\hat{v}_1''$ とすると、遷移 $\hat{v}_1 \xrightarrow{w'} \hat{v}_1''$ のあと、 $\hat{v}_1''$ から系列 $\gamma \in T(\hat{v}_1)$ による遷移の行き先は上と同一の節点 $\hat{v}_1'$ である。また、 $\hat{v}_1'$ におけるスタックの内容は同じゆえ、 $w_1, w', \gamma \in T(\hat{v}_1)$ がなりたつ。また、 $T(\hat{v}_1) = L(\bar{v}_1, \bar{v}_E)$ より、 $w_1, w', \gamma \in L(\bar{v}_1, \bar{v}_E)$ がなりたつ、 $\bar{v}_1 \xrightarrow{w_1, w'} \bar{v}_1''$ より、 $\gamma \in L(\bar{v}_1'', \bar{v}_E)$ となる。こうして、 $L(\bar{v}_1', \bar{v}_E) \subset L(\bar{v}_1'', \bar{v}_E)$ が言える。逆の関係も同様にして言え、 $L(\bar{v}_1', \bar{v}_E) = L(\bar{v}_1'', \bar{v}_E)$ となる (証明終)

以下、 $\bar{v}_1$ から $T(\hat{v}_1)$ 内の任意の系列による遷移の行き先の節点を $\bar{v}_1'$ と書く。

(iii) $\bar{v}_1$ から $T(\hat{v}_1)$ 内の系列によって遷移したとき、途中で、節点 $\bar{v}_1'$ を通ることはない (図 3-6 参照)

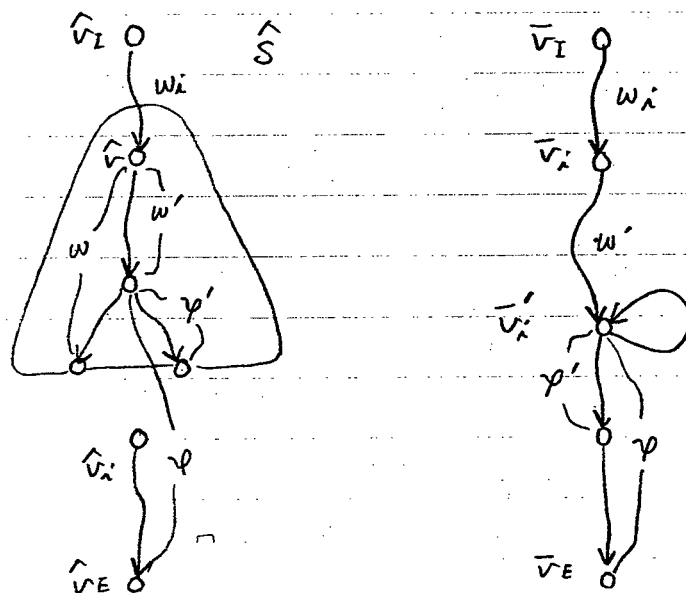


図 3-6 補題 3-4-iii の証明の説明図

(iii) の証明) $\bar{v}_i$ から $T(\hat{G})$ 内のある系列 w の付いた道を遷移したとき, w の真の始系列 w' が付いた道を遷移した所で節点 $\bar{v}_i$ に至ったと仮定する。 $\bar{S}$ は条件 A_1 を満足していることから, 節点 $\bar{v}_i$ から途中で $\bar{v}_i$ を通ることなく $\bar{v}_E$ に至る道が存在する。その道に付けられた系列を γ とすると, $w, w', \gamma \in L(\bar{v}_i, \bar{v}_E) = T(\hat{G}_E)$ となる。今において, w' が $w \in T(\hat{G})$ の始系列であるため, w', γ のある始系列 w'' (γ' は空系列でない) が $T(\hat{G})$ に含まれる。すると, (ii) より, $\bar{v}_i \in \mathcal{O}(\hat{G})$ から系列 w'' が付いた行き先が $\bar{v}'$ となる。 $\bar{v}_i \xrightarrow{w} \bar{v}_i'$ より, $\bar{v}_i'$ から γ の始系列 γ' による遷移の行き先が $\bar{v}_i'$ となる。これは, γ の定義に反する。 (証明終)

(iv) ある k_i に対して, $\bar{v}_i' = E^{k_i}(\bar{v}_i)$ がなりたつ。

(iv) の証明) ①, $\bar{v}_i$ から $\bar{v}_i'$ を通ることなく $\bar{v}_E$ に行く遷移 $\bar{v}_i \xrightarrow{w} \bar{v}_E$ が存在したと仮定する。 $w, w \in L(\bar{v}_i, \bar{v}_E) = T(\hat{G}_E)$ より, 今において, w のある始系列 w' が $T(\hat{G})$ 内に存在する。すると, (ii) より, $\bar{v}_i \xrightarrow{w} \bar{v}_i'$ となるが, これは, 遷移 $\bar{v}_i \xrightarrow{w} \bar{v}_E$ の定義に反する。 (証明終)

つぎに, $T(\hat{G}) \cap L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ であることを以下の (v), (vi) を述べて証明する。

(v) $\bar{S}$ は条件 A_1 を満足しており, “無駄な” 節点を含んでなく, また, $\bar{S}$ と $\hat{G}$ は等価である。ゆえに, 任意の $w \in L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ に対して, 今において, 遷移 $\hat{G}_E \xrightarrow{w} \hat{G}_i$ ($\hat{G}_E \xrightarrow{w} \bar{v}_i$, $\bar{v}_i \in \mathcal{O}(\hat{G}_i)$) のあとに, $\hat{G}_i$ から系列 w が付いた遷移が存在する。

(vi) $\bar{S}$ における任意の $w \in L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ に対して, 今において, $w \in T(\hat{G})$ がなりたつ。

(vi) の証明) ある $w \in L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ に対して, w にある (空でない) 系列 w' を付けた $w w'$ が $T(\hat{G})$ 内

に含まれるか、あるいは、 w のある真の始系列 w'' が $T(\psi)$ 内に含まれるかのいずれかである。前者の場合、 $T(\psi) \subset L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ であることから、 $ww' \in L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ になりたつ。しかし、 $w \in L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ でもあることから (iii) と矛盾する。同様の方法で証明できる。

(証明終)

§ 4.1 序言

プログラムをサブルーチンなどのモジュールに分割し、構造的に書くことは、プログラムのドキュメント、コーディング、デバッグ、改良などの立場から重要なことである。プログラムをモジュールに分割するとき、モジュール内の命令の機能的関連性を高め、また、モジュール間については、パラメータ（とくに判定結果のフラッグ）や共用データなどによるお互いの影響ができるだけ少なくなるように分割することが望ましい。

Stevens らは論文 “Structured design⁽¹⁾” において、プログラムモジュール間の影響範囲をつぎのように局限するというプログラムの一つの設計法を述べている。モジュール⁵における判定結果（たとえば、入力端末装置が正常に動作しているかという判定結果）がモジュール内におけるコントロールの流れ（たとえば、オペレータに通報する必要があるかないか）に影響を与えるとき、その判定からたへ“エフェクト”があるといい、その判定からエフェクトがあるようなモジュールの集合をその判定の“エフェクトの範囲 (scope of effect)” と呼ぶ。

Stevens らは、“エフェクトの範囲”がその判定を含むモジュール自身またはそのモジュールから（つきつぎ）コールされて至るようなモジュールの範囲（“制御の範囲 (scope of control)” といわれる）に完全に含まれることが望ましいと考え、プログラムの設計段階において、もとのモジュールの合併（もとのいくつかのモジュールを合併しあらたに一つのモジュールをつくること）、および、もとのコール関係の変更（たとえば、もとではコー

ル関係が全然無かったモジュールを自分のサブルーチンとして持ってくること)という手段により、“エフェクトの範囲”が“制御の範囲”にあさまのようにモジュール群を再構成するということを提案している。

さて、ここでは、Stevensらと異なって、もとのコール関係は変更しないことにし(これによりプログラムの修正が少なく済む)、モジュールの合併のみを行なうモジュール群の再構成法を考える。コール関係は保存する代り、“エフェクトの範囲”を、Stevensらのように“制御の範囲”(いわゆる“子孫”のモジュール)だけに限らず、いわゆる“親”のモジュール(そのモジュールを直接コールするモジュール)と“兄弟”のモジュール(“親”のモジュールが直接コールするモジュール)にまで許すことにする。その方が、合併によってあらたにできたモジュールが大きくなりすぎる可能性が少なく、また、“エフェクトの範囲”をそこまで許していわゆる“Structured design”における構造の規則性(“well formed”性)の立場から不都合さはそれ程無いであろう。この章では、合併をできるだけせずに、上記の条件を満たすように与えられたモジュール群を再構成することを考えるが、議論を簡単にするため、具体的には、できたあとのモジュールの個数が最大となるような再構成法について考察する。

各モジュールに対し、それを表わす一つの節点 v を設け、 v に対応するモジュールが w に対応するモジュールをコールすることがあれば v から w へ有向枝を引き、各モジュール間のコール関係を有向グラフ $G = (V, A)$ (V , A はそれぞれ節点および有向枝の集合)で表現する。ここでは、再帰的なコールは考えないので、 G は非サイクリックグラフになる。

モジュール s 内の判定からモジュール t へ “エフェクト” があるとき、節点对 (s, t) を “エフェクト枝” と呼び、 “エフェクト” 関係の全体はそのようなエフェクト枝の集合 E で与えられる。

一つのモジュールにまとめられるべき互いに合併されるものモジュールの集合をそれぞれ “ブロック” と呼ぶことにする。この章では、コール関係のグラフ G と上述のようなエフェクト枝の集合 E が与えられたとき、 “エフェクトの範囲” が前述の範囲におさまるように、各モジュールの全体を最大個数のブロックに分割するためのアルゴリズムについて考察し、 G をとくに有向木に限った場合、そのブロック数最大の分割が $|V| + |E| \times G(|V|)^+$ のオーダーの手数で能率よく求められることを示す。4.2 節で定義と問題について詳しく述べたあと、4.3 節でそのアルゴリズムを述べる。

⁺ $|X|$ は集合 X の元の個数を表わす。関数 $G(n)$ は、 $F(i)$ を $F(0) = 1$, $F(i) = 2^{F(i-1)}$ と定義したとき、 $F(k) \geq n$ であるような最小の k を表わす。 $G(n)$ はきわめてゆっくり増加する関数である。(13)

§ 4.2 定義と問題

まえがきで述べたように、プログラムモジュール間のコール関係を有向グラフ $G = (V, A)$ で表現される。 V は節点の集合、 A は有向枝の集合である。節点 v から w へ有向枝があることを $v \rightarrow w$ と書く。グラフ G の二つの節点 s, t に対し、記法 $s \rightarrow^+ t$ は、 s から t へ枝の向きに沿ったパスが存在することと表わす (s, t が同一節点のときはつねに $s \rightarrow^+ t$ が成り立つとする)。また、便宜上、 $s \rightarrow^+ t$ でパスそのものを指すときもある。

本論文では、コール関係を表現するグラフ G は、つぎの条件を満たしているものとする。

(イ) G には主モジュールに対応する節点 v_M が一つ存在し、 G 内の各節点 v に対し $v_M \rightarrow^+ v$ である (すなわち、 v_M からその節点へ至るパスがある)。

(ロ) G 内にはループ (枝の向きに沿った閉路⁺) が存在しない。

エフェクト関係の全体はエフェクト枝の集合 E で表わされる。 G において s から t へ至るような向きを δ フォパスが存在するようなエフェクト枝 (s, t) は我々の問題において全く考慮する必要がない (なぜなら、 t を含むあらたなモジュールは s を含むあらたなモジュールは s を含むあらたなモジュールの“制御の範囲”内に必ずある) ので、

(ハ) E には、 G において $s \rightarrow^+ t$ であるようなエフェクト枝 (s, t) は含まれていないと考える。

⁺ 長さ 1 以上のパス $v \rightarrow^+ v$ のこと。パスの長さとはそれに含まれる枝の個数。

さて、節点集合 V の分割 $\pi = \{\beta_1, \beta_2, \dots, \beta_l\}^+$ を考える。 V の部分集合である各 β_i をブロックと呼ぶ。 $\beta_i \neq \beta_j$ で、かつ $v \rightarrow w$, $v \in \beta_i$, $w \in \beta_j$ なる節点 v , w が存在するとき、 $\beta_i \Rightarrow \beta_j$ と書く。

この章ではつきの問題を考察する。

[問題 4-1] コール関係のグラフ G , エフェクト枝の集合 E が与えられたとき、以下の三つの条件を満足するブロック数最大の分割を一つ求めよ。

[連結条件] 同一ブロックに属する任意の二つの節点に対し、そのブロック内の節点間の有向枝を無向枝と見なしたとき、一方の節点から他方の節点へ至るこれらの無向枝のみでできるパスが存在する。

[非サイクルブロック条件] ブロック間の関係 $\Rightarrow$ によって閉路 ($\beta_i \Rightarrow \dots \Rightarrow \beta_i$) をつくることはない。

[近接条件] E に属する各エフェクト枝 (s, t) に対し、 s, t を含むブロックをそれぞれ β_i, β_j とすると、 β_i と β_j の間には、以下の (a) ~ (d) の少なくとも一つが成り立つ。

(a) $\beta_i = \beta_j$

(b) $\beta_i \Rightarrow \beta_j$

(c) $\beta_j \Rightarrow \beta_i$

(d) あるブロック β_k が存在し、 $\beta_k \Rightarrow \beta_i$ かつ $\beta_k \Rightarrow \beta_j$,
ただし、 $\beta_i \neq \beta_j$

まえがきで述べたように、この問題は、“制御の範囲”内へ向いていないエフェクト枝による好ましくないエフェクト関係を自分自身か、“親子”間か、または“兄弟”間のみにも局限されるように、もとのエジュールを合併し、エジュール群を再構成する問題である。

+ 各 β_i は V の空 ($\emptyset$) でない部分集合であり、 $\beta_1 \cup \beta_2 \cup \dots \cup \beta_l = V$, $i \neq j$ ならば $\beta_i \cap \beta_j = \emptyset$.

§ 4.3

G が有向木である場合

ここでは、与えられたコール関係のグラフ G がとくに有向木である場合、前述の3条件を満たすブロック数最大の分割が、 $|V| + |E| \times G(|V|)$ のオーダの手数で（ランダムアクセス機械⁽¹²⁾により）求められることを示す。

4.3.1 カット枝集合を求める問題への変換

G が有向木である場合、分割 π が与えられた G に対し「連続条件を満たしていれば、 π は自動的に「非サイクリック条件」も満足する。また、 G が有向木である場合、「連続条件」を満足している分割 π を指定するには、ブロック間の（もとの G における）枝の集合を与えてよい。このような枝を、 G をいくつかのブロックに切断するという意味で、“カット枝”と呼ぶ。この4.3節では、以下、このようなカット枝の集合を求めるという立場で議論する。

与えられたエフェクト関係 E に属する節点对 (s, t) は、前述の仮定 (11) より、 $t \rightarrow s$ の場合、そうでない場合により、図4-1の (a) または (b) の形になる。(b)

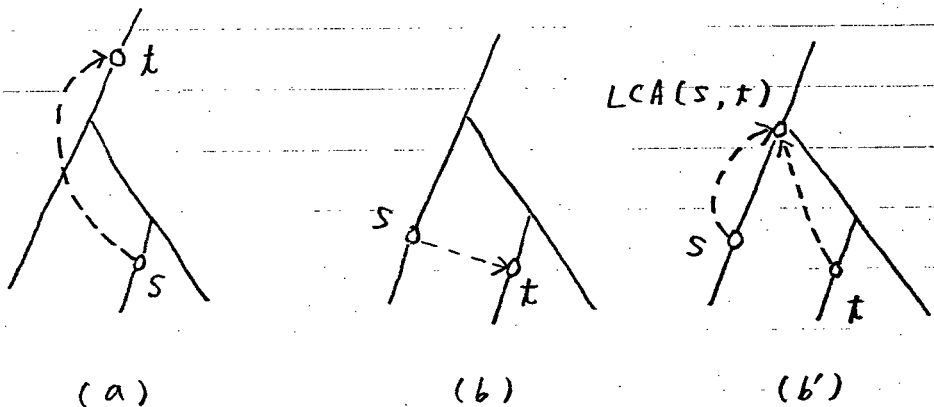


図 4-1

E および $f(E)$ の枝

の形のエフェクト枝を同図 (b') に示されているような二つのエフェクト枝で置換えることを考える。E における (a) の形のエフェクト枝はそのままに、(b) の形のエフェクト枝を、それぞれ、(b') の形の二つのエフェクト枝で置き換えて得られるエフェクト枝の集合を $f(E)$ とする。形式的には、 $f(E)$ はつぎのように定義される。任意の二つの節点 v_1, v_2 に対し、 $v \rightarrow v_1, v \rightarrow v_2$ であり、かつ、 $v \rightarrow v', v' \rightarrow v_1, v' \rightarrow v_2$ なる $v' (\neq v)$ が存在しないとき、 v を v_1, v_2 の "lowest common ancestor" と呼び、 $LCA(v_1, v_2)$ で表わす (図 4-1 (b'))。

[$f(E)$ の定義] $LCA(s, t) = t$ であるような (言い換えれば、 $t \rightarrow s$ であるような) エフェクト枝 $(s, t) \in E$ については、 $(s, t) \in f(E)$ とする。 $LCA(s, t) = t$ でないような (E の仮定 (ハ) より、 $LCA(s, t) = s$ となることはない) エフェクト枝 $(s, t) \in E$ については、 $(s, LCA(s, t)) \in f(E), (t, LCA(s, t)) \in f(E)$ とする。

一般に、エフェクト関係 E で、 E に属する任意のエフェクト枝 (s, t) に対し $t \rightarrow s$ が成り立つようなエフェクト関係 E のクラスを $\mathbb{E} \text{ backwards}$ と書く。もちろん、任意の E に対し、 $f(E) \in \mathbb{E} \text{ backwards}$ である。

コール関係の有向木 G とエフェクト枝の集合 $B \in \mathbb{E} \text{ backwards}$ が与えられたとき、つぎの問題を考える。

[問題 4-2] 有向木 G と $\mathbb{E} \text{ backwards}$ に属するエフェクト関係 B が与えられたとき、以下に述べる「カット条件」を満足する枝 (カット枝) の集合 Δ で、枝の個数が最大であるものを一つ求めよ。

[カット条件] $B (\in \mathbb{E} \text{ backwards})$ 内の任意のエフェクト枝 (s, t) に対し、 G 上のパス $t \rightarrow s$ が Δ 内の枝をたかだか 1 個しか含まない。

有向木 G と一般のエフェクト関係 E が与えられたときの
前述の問題 4-1 は、つぎの補題 4-1 に示すように、 $B = f(E)$ とおいた上記の問題 4-2 に帰着される

〔補題 4-1〕 有向木 G とエフェクト関係 E が与えられたとき、 G と $f(E)$ に対し〔カット条件〕を満足する枝の集合で、枝数最大のもの（すなわち、 $B = f(E)$ とおいた問題 4-2 の解）を Δ とすると、 Δ できる分割元は、 G と E に対し〔連結条件〕、〔非サイクリック条件〕、〔近接条件〕を満足する G の節点の分割で、ブロック数最大のもの（すなわち、問題 4-1 の解）である。

（証明） G と $f(E)$ に対し〔カット条件〕を満足するカット枝の集合を Δ とし、 Δ できる分割を π とする。
 $(s, t) \in E$ かつ $(s, t) \in f(E)$ であるようなエフェクト枝 (s, t) に対し、 Δ が〔カット条件〕を満足することから、パス $t \rightarrow s$ は Δ 中のカット枝をたかだか 1 個しか含まないが、これは、 s, t を含む π のブロックをそれぞれ β_i, β_j とすると $\beta_i = \beta_j$ または $\beta_j \Rightarrow \beta_i$ であることを意味する。
 $(s, t) \in E$ かつ $(s, \text{LCA}(s, t)) \in f(E)$ 、 $(t, \text{LCA}(s, t)) \in f(E)$ であるような E のエフェクト枝 (s, t) に対し、 Δ が〔カット条件〕を満足することから、 $s, t, \text{LCA}(s, t)$ を含む π のブロックをそれぞれ $\beta_i, \beta_j, \beta_k$ とすると、 β_i に関し $\beta_i = \beta_k$ または $\beta_k \Rightarrow \beta_i$ 、 β_j に関し $\beta_j = \beta_k$ または $\beta_k \Rightarrow \beta_j$ が成り立つ。
上記のいずれの場合でも、 β_i と β_j について〔近接条件〕中の (a) ~ (d) の少なくとも一つが成り立ち、 π は〔近接条件〕を満たす。

さて、逆に、前述の 3 条件を満たす分割を π' とし、 π' の各ブロックを区切るカット枝の集合を Δ' とする。 $(s, t) \in E$ かつ $(s, t) \in f(E)$ であるようなエフェクト枝 (s, t) に対し、 π' が〔近接条件〕を満足することから、 s, t

を含む π' のブロックをそれぞれ β_i, β_j とすると $\beta_i = \beta_j$ または $\beta_j \Rightarrow \beta_i$ が成り立つが、これは、ノス $x \rightarrow s$ が Δ' 中のカット枝をたかだか1個しか含まないことを意味する。 $(s, x) \in E$ か $(s, LCA(s, x)) \in f(E)$, $(x, LCA(s, x)) \in f(E)$ であるような E のエフェクト枝 (s, x) に対し、 π' が [近接条件] を満足することから、 s, x を含む π' のブロックをそれぞれ β_i, β_j とすると、前述の [近接条件] 中の (a) ~ (d) の一つがなりたつ。
 $LCA(s, x)$ を含む π' のブロックを β_k とする。(a) $\beta_i = \beta_j$ の場合、 π' が [連結条件] を満足することより、ブロック β_i, β_j はともに $LCA(s, x)$ を含まねばならず、 $\beta_k = \beta_i = \beta_j$ となる。(b) $\beta_i \Rightarrow \beta_j$ の場合、 β_i は $LCA(s, x)$ を含まねばならず、 $\beta_k = \beta_i$ となる。(c) $\beta_j \Rightarrow \beta_i$ の場合は、 $\beta_k = \beta_j$ となる。(d) $\beta_i \neq \beta_j$ で、ある β_h が存在して、 $\beta_h \Rightarrow \beta_i, \beta_h \Rightarrow \beta_j$ の場合は、 β_h は $LCA(s, x)$ を含まねばならず $\beta_k = \beta_h$ でなければならぬ。以上のことより、ノス $LCA(s, x) \rightarrow s, LCA(s, x) \rightarrow x$ は、それぞれ、 Δ' 中のカット枝をたかだか1個しか含まない。すなわち、 Δ' は [カット条件] を満たすことになる。

また、上記の変換において、 $|\pi| = |\Delta| + 1, |\Delta'| = |\pi'| - 1$ のように個数が対応しているので補題 4-1 が成り立つ。
 (証明終)

この補題 4-1 より、有向木 G とエフェクト関係 E が与えられたとき問題 4-1 の解を得るには、まず、 G, E から $f(E)$ を求め、つぎに、 G と $f(E)$ に対し問題 4-2 の解 Δ を求めればよいことがわかった。 Δ から節点の分割元は前述の意味で自動的に定まる。

さて、与えられた G と E から、 $f(E)$ は、文献 (13) の "lowest common ancestor" を求めるアルゴリズムを利用して容易に求められる。

〔手続き 4-1: E から $f(E)$ への変換〕 E に属するすべての節点对 (s, t) に対し, 文献 (13) のアルゴリズムにより $LC A(s, t)$ を求め, それを用いて, $f(E)$ の定義のように新しいエフェクト枝を構成する。

〔補題 4-2〕 手続き 4-1 の手数は $|V| + |E| \times G(|V|)$ のオーダーである。

(証明) 文献 (13) のアルゴリズムの手数は $|V| + |A| + |E| \times G(|A|)$, 新しいエフェクト枝を構成する手数は $|E|$ のオーダーであることからあきらか (G が有向木ゆえ $|V|$ と $|A|$ のオーダーは等しい)。(証明終)

4.3.2 最大カット枝集合を求める手順の説明

ここでは, 有向木 G とエフェクト関係 $B \in \mathcal{E} \text{ backwards}$ が与えられたとき, 前述の [カット条件] を満たすカット枝の集合で, 枝数が最大のものを一つ (それを Δ とする) 求める方法の概略を述べる。

有向木 G の各節点 v の“深さ”を根 v_M からのパス $v_M \rightarrow v$ 上の枝の個数とし, $DEPTH(v)$ で表わす。もちろん, $v \rightarrow w$, $v \neq w$ ならば, $DEPTH(v) < DEPTH(w)$ である。 G の各節点にこの値を割りふるには, いわゆる “depth first search⁽¹²⁾” の方法により G の枝数 (したがって節点数) に比例する手数で容易にできる。

さて, その Δ を求める方法というのは, 直感的には, 葉の方から根の方に向かって進みながら, ある枝 (v, w) より下位の部分木全体についてカット枝の集合が求まっているとき, その枝 (v, w) をカット枝にしても [カット条件] に反しないならそれをカット枝に選び, できるだけ葉に近い方で区切るようにしている。それをいわゆる depth first の手順で書けば, つぎのようになる。

[Δ を求める手順]

1. G の各節点 v に対し, $\Delta(v) \leftarrow \emptyset$ とする. ($\Delta(v)$ には A の任意の部分集合が入る)
2. 以下に定義する再帰的手続き $CUTEDGE$ の実行パラメータを根 v_M にして実行する. $\Delta(v_M)$ が求まる.
3. $\Delta \leftarrow \Delta(v_M)$

procedure $CUTEDGE(v)$ (節点 v 以下の部分木に対するカット枝の集合 $\Delta(v)$ を求める手続き)

1. v が葉であれば, 何もしずリターンする.
2. v が葉でないとき,
 $v \rightarrow w$ であるような各節点 w に対し, つぎの 2.1 ~ 2.3 を実行する.
 - 2.1 $CUTEDGE(w)$ を実行する.
 - 2.2 もし枝 (v, w) をカット枝に選んでも現在の時点では [カット条件] に反しないとき,
 $\Delta(v) \leftarrow \Delta(v) \cup \Delta(w) \cup \{(v, w)\}$
 - 2.3 そうでないとき (選ぶと現在の時点ですでに [カット条件] に反するとき)
 $\Delta(v) \leftarrow \Delta(v) \cup \Delta(w)$
3. $v \rightarrow w$ であるようなすべての w について上の操作が終ったとき, リターンする.

この手順により, 目的のものが正しく求まるという証明に入る前に, その証明, および手順中の 2.2, 2.3 における “枝 (v, w) をカット枝に選んでも [カット条件] に反しないかどうか” という判定法にも使える “CONTAIN” と “UNCONTAIN” という概念を導入しよう.

G と B は固定して考える. G の任意の節点 w について, w を根とする G の部分木を G_w と表わす.

G の節点 w と G_w に属する枝の部分集合 Δ' (ただし, Δ' を G の枝の部分集合とみたとき, G と B に対する [カ

ット条件] を満たしているとする⁺。これを単に Δ' が [カット条件] を満たしているという) に対し, $\text{CONTAIN}(w, \Delta')$ を, δ 1 $t \rightarrow w$, $w \rightarrow s$ かつ $w \rightarrow s$ が Δ' 内の枝を含む⁺⁺, というようなエフェクト枝 (s, t) が B に存在するなら, そのような節点 t の深さの最小値, 存在しなければ w の深さと定義し, $\text{UNCONTAIN}(w, \Delta')$ を, δ 1 $t \rightarrow w$, $w \rightarrow s$ かつ $w \rightarrow s$ が Δ' 内の枝を含まない, というようなエフェクト枝 (s, t) が B に存在するなら, そのような節点 t の深さの最小値, 存在しなければ w の深さと定義する (図 4-2 参照)

G において $v \rightarrow w$ とし, 今, Δ_v を [カット条件] を満たしている G_v 内の枝の部分集合とするとき, Δ_v のうち G_w に属する枝のみでできる集合を Δ_w と書くことにする。定義より, 以下の式が成り立つ (図 4-3 参照)

$$\begin{aligned} & \text{CONTAIN}(v, \Delta_v) \\ &= \text{MIN} [\{ \text{CONTAIN}(w, \Delta_w) \mid v \rightarrow w, (v, w) \notin \Delta_v \} \\ & \quad \cup \{ \text{UNCONTAIN}(w', \Delta_{w'}) \mid v \rightarrow w', (v, w') \in \Delta_v \} \\ & \quad \cup \{ \text{DEPTH}(v) \}] \quad (1) \end{aligned}$$

$$\begin{aligned} & \text{UNCONTAIN}(v, \Delta_v) \\ &= \text{MIN} [\{ \text{UNCONTAIN}(w, \Delta_w) \mid v \rightarrow w, (v, w) \notin \Delta_v \} \\ & \quad \cup \{ \text{DEPTH}(t) \mid (v, t) \in B \} \\ & \quad \cup \{ \text{DEPTH}(v) \}] \quad (2) \end{aligned}$$

準備ができたので, 前述の Δ を求める手順の正しさの証明に入ろう。つぎの補題 4-3 を示せば十分である。

⁺ 以下では, [カット条件] を満たしているような Δ' に対してのみ, CONTAIN や UNCONTAIN を使用している。

⁺⁺ Δ' が [カット条件] を満たしているので, 含むとしても高々 1 個である

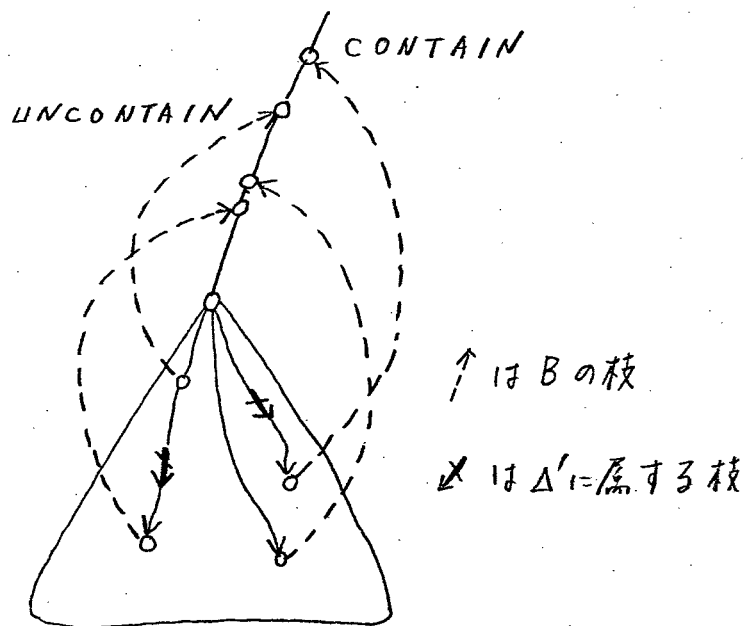


図 4-2 CONTAIN, UNCONTAIN の定義

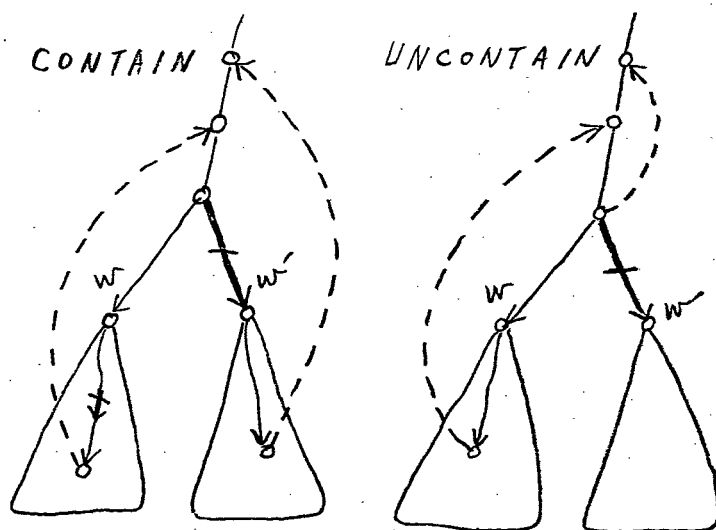


図 4-3 CONTAIN, UNCONTAIN の性質

[補題 4-3] 前述の Δ を求める手順における手続き $CUT\ EDGE(v)$ によって求められた G_v 内の枝の部分集合 $\Delta(v)$ は, G_v 内の枝の部分集合で「カット条件」を満たす枝数が最大のものであり, さらに, そのうちで $CONTAIN$ の値が最大のものである.

(証明) 帰納法を用いて証明する.

I. v が葉である場合, G_v に属する枝はないので, $\Delta(v) = \emptyset$ は, 「カット条件」を満たす枝数最大のもののうち, $CONTAIN$ の値最大のものであることはあきらか.

II. v が葉でない場合, 帰納法の仮定として, $v \rightarrow w$ である各節点 w に対し $\Delta(w)$ が「カット条件」を満たす枝数最大のもののうち, $CONTAIN$ の値最大のものであると仮定する.

$\Delta(v)$ が「カット条件」を満たすことは, 手続き $CUT\ EDGE(v)$ より, あきらかである.

$\Delta(v)$ が「カット条件」を満たすもののうち, 枝数が最大のものであることの証明は背理法で行う. そのために, G_v 内の枝のみからなるカット枝の集合 Δ_v で, (イ) 「カット条件」を満足し, かつ $|\Delta_v| > |\Delta(v)|$ なるもの, または, (ロ) 「カット条件」を満足し, かつ $|\Delta_v| = |\Delta(v)|$ であるが, $CONTAIN(v, \Delta_v) > CONTAIN(v, \Delta(v))$ なるものが存在すると仮定する.

$v \rightarrow w$ なる w に対し, Δ_v のうち G_w 内に属するカット枝集合 Δ_w で表わす. Δ_v が「カット条件」を満たしていれば, Δ_w も「カット条件」を満たすことはあきらかである.

(イ) の場合:

帰納法の仮定より, $v \rightarrow w$ なる各 w に対し, $|\Delta_w| \leq |\Delta(w)|$ ゆえ, $|\Delta_v| > |\Delta(v)|$ であるためには, 以下の

命題 P が成り立たねばならない。

[命題 P : $v \rightarrow w$ なる w のなかに, $|\Delta w'| = |\Delta(w')|$ であり, かつ $(v, w') \in \Delta v$, $(v, w') \notin \Delta(v)$ であるような w' が存在する.]

しかし, この命題は, 以下のようにして, $|\Delta(w')|$ の CONTAIN の値が最大という帰納法の仮定に矛盾する. $\Delta w'$ は [カット条件] を満足し, また $\Delta(w')$ に関する帰納法の仮定 ([カット条件] を満たすうちで枝数最大) と $|\Delta w'| = |\Delta(w')|$ より, $\Delta w'$ も [カット条件] を満たすうちで枝数最大のものである. 一方 $(v, w') \in \Delta v$ で, かつ Δv が [カット条件] を満たしているので, CONTAIN の定義より, $\text{CONTAIN}(w', \Delta w') = \text{DEPTH}(w')$. また, $(v, w') \notin \Delta(v)$ であることは, 前述の手順における $\Delta(v)$ の求め方および CONTAIN の定義より, $\text{CONTAIN}(w', \Delta(w')) < \text{DEPTH}(w')$ であったことを意味する. ゆえに, $\text{CONTAIN}(w', \Delta(w')) < \text{CONTAIN}(w', \Delta w')$ となる. これは, $\Delta(w')$ が [カット条件] を満たす枝数最大のもののうち, CONTAIN の値が最大のものであるという帰納法の仮定に反する.

(ロ) の場合:

CONTAIN の定義 (または式 (1)) より, $\text{CONTAIN}(v, \Delta(v))$ の値は, (a) $(v, w) \notin \Delta(v)$ であるようなある w に対する $\text{CONTAIN}(w, \Delta(w))$ か, (b) $(v, w) \in \Delta(v)$ であるようなある w に対する $\text{UNCONTAIN}(w, \Delta(w))$ か, (c) $\text{DEPTH}(v)$ かのいずれかと等しくなる.

(a) の場合: その w に対し, $\exists (v, w) \in \Delta v$ と仮定すると, $\text{CONTAIN}(v, \Delta v) \leq \min\{\text{DEPTH}(x) \mid (s, x) \in B, x \rightarrow v, w \rightarrow s\} \leq \text{CONTAIN}(w, \Delta(w))$ となり, 一方, (ロ) および (a) の場合の条件より $\text{CONTAIN}(v, \Delta v) > \text{CONTAIN}(v, \Delta(v)) = \text{CONTAIN}(w, \Delta(w))$ であるが,

この二つの式は矛盾するので、 $(v, w) \in \Delta_v$ でなければならない。

さて、帰納法の仮定より $|\Delta_w| \leq |\Delta(w)|$ であるが、これをさらに二つの場合に分けて考える。

(a, 1) $|\Delta_w| < |\Delta(w)|$ の場合、(a) の場合の条件と上記の結果より $(v, w) \in \Delta(v)$ 、 $(v, w) \in \Delta_v$ であり、(ロ) の場合の条件の $|\Delta_v| = |\Delta(v)|$ であるためには、(イ) の場合と同じく、命題 P が成り立つ必要がある。しかし、(イ) の場合と同様、これは帰納法の仮定に矛盾する。

(a, 2) $|\Delta_w| = |\Delta(w)|$ の場合、 Δ_w も $(\Delta(w))$ と同じく「カット条件」を満たすうちで枝数最大となる。一方、(ロ) および (a) の場合の条件より $\text{CONTAIN}(v, \Delta_v) > \text{CONTAIN}(v, \Delta(v)) = \text{CONTAIN}(w, \Delta(w))$ であるが、一般に CONTAIN の定義より $\text{CONTAIN}(w, \Delta_w) \geq \text{CONTAIN}(v, \Delta_v)$ ゆえ、 $\text{CONTAIN}(w, \Delta_w) > \text{CONTAIN}(w, \Delta(w))$ となる。これは $\Delta(w)$ が「カット条件」を満たす枝数最大のもののうち、 CONTAIN の値が最大という帰納法の仮定に反する。

(b) の場合：その w に対し、 $\forall (v, w) \in \Delta_v$ と仮定すると、 $\text{CONTAIN}(v, \Delta_v) \leq \min \{ \text{DEPTH}(x) \mid (s, x) \in B, x \rightarrow v, w \rightarrow s \} \leq \text{UNCONTAIN}(w, \Delta(w))$ となり、一方、(ロ) および (b) の場合の条件より $\text{CONTAIN}(v, \Delta_v) > \text{CONTAIN}(v, \Delta(v)) = \text{UNCONTAIN}(w, \Delta(w))$ であるが、この二つの式は矛盾するので、 $(v, w) \in \Delta_v$ でなければならない。

さて、帰納法の仮定より $|\Delta_w| \leq |\Delta(w)|$ であるが、(b) の場合の条件と上記の結果より、 $(v, w) \in \Delta(v)$ 、 $(v, w) \in \Delta_v$ であり、(ロ) の場合の条件の $|\Delta_v| = |\Delta(v)|$ であるためには、(イ) の場合と同じく、命題 P が成り立つ必要がある。しかし、(イ) の場合と同様、これは帰納法の仮

定に矛盾する。

(C) の場合： この場合の条件 $CONTAIN(v, \Delta(v)) = DEPTH(v)$ は、 $CONTAIN$ の定義および (D) の場合の条件より得られる式 $DEPTH(v) \geq CONTAIN(v, \Delta(v)) > CONTAIN(v, \Delta(v))$ に矛盾する。 (証明終)

補題 4-3 より、前述の手続き $CUTEDGE(v)$ 中の $CUTEDGE(w)$ によって求められた $\Delta(w)$ は [カット条件] を満たしている。したがって、その手続き中の 2.2, 2.3 における “枝 (v, w) をカット枝に選んでも [カット条件] に反しないか” のテストは、 $CONTAIN$ の定義より、 $CONTAIN(w, \Delta(w))$ を用いて、

$CONTAIN(w, \Delta(w)) = DEPTH(w)$ のとき、かつそのときのみ、 (v, w) をカット枝に選んでも [カット条件] に反しない。

のように行える。また、 $\Delta(w)$ は [カット条件] を満たしているので、 $CONTAIN(v, \Delta(v))$ の値は、 $\Delta(v)$ を求めるとき、前述の式 (1), (2) によってつきつきと求めていくことができる。

4.3.3 最大カット枝集合を求めるアルゴリズム

4.3.2 節で述べた $\Delta(v)$ を求める手続き $CUTEDGE(v)$ を $CONTAIN(w, \Delta(w))$ および $UNCONTAIN(w, \Delta(w))$ がわかっているとして、それを用いた枝 (v, w) をカット枝に選ぶかどうかの判定法と $CONTAIN(v, \Delta(v))$, $UNCONTAIN(v, \Delta(v))$ の求め方を追加して、完全なものに書き直し、それを [手続き 4-2] として図 4-4 に示す。

そのアルゴリズムでは、各節点 v に対し、 $\Delta(v)$, $CONTAIN(v, \Delta(v))$, $UNCONTAIN(v, \Delta(v))$ の初期値として、それら $\emptyset$, $DEPTH(v)$, $\min[\{DEPTH(x) \mid (v, x) \in B\}] \cup$

$\{ \text{DEPTH}(v) \}$ をセットしておき、再帰の手続き $\text{CUT_EDGE}(v)$ 内において、 $\Delta(v)$ 、 $\text{CONTAIN}(v, \Delta(v))$ 、 $\text{UNCONTAIN}(v, \Delta(v))$ の値が正しく求まったとき、リターンする。

アルゴリズム中の $\Delta(v)$ は枝の集合を表わしているが、ポインタでつながるようにすれば $\Delta(v) \leftarrow \Delta(v) \cup \Delta(w) \cup \{(v, w)\}$ などは一定時間で処理できる。アルゴリズムの初期設定に $|V| + |B|$ のオーダの枝数を要し、また、各節点 v に対し、 $\text{CUT_EDGE}(v)$ は一回しかコールされず、一つの節点に対しては一定の枝数で済む。したがって、補題 4-4 が成り立つ。

[補題 4-4] 図 4-4 の手続き 4-2 により、 G 、 B に対して、[カット条件] を満たす最大カット枝集合が $|V| + |B|$ のオーダの枝数で求まる。

手続き 4-1 と手続き 4-2 をあわせることにより、この章の主要な結果であるつぎの定理が得られる。($f(E)$ の枝数は m との E のそれのたかだか 2 倍)

[定理 4-1] 与えられたコール関係のグラフ $G = (V, A)$ 、エフェクト枝の集合 E に対し、 G が有向木であれば、問題 4-1 の解であるブロック数最大の分割が $|V| + |E| \times G(|V|)$ のオーダの枝数で求められる。

input : 有向木 $G = (V, A)$, エッジト枝の集合 $B \in \mathcal{E}^{backwards}$
output : [カット条件] を満たす最大カット枝集合 Δ

algorithm :

```

begin
  for each  $v \in V$ 
    begin
       $\Delta(v) \leftarrow \phi$  ;
       $\text{CONTAIN}(v, \Delta(v)) \leftarrow \text{DEPTH}(v)$  ;
       $\text{UNCONTAIN}(v, \Delta(v)) \leftarrow \min[\{\text{DEPTH}(t) \mid (v, t) \in B\} \cup \{\text{DEPTH}(v)\}]$  ;
    end
  call CUTEDGE( $v_M$ ) ;
   $\Delta \leftarrow \Delta \cup \Delta(v_M)$  ;
end

procedure CUTEDGE( $v$ )
begin
  if  $v$  is a leaf then
    return ;
  else
    begin
      for each  $w$  such that  $(v, w) \in A$ 
        begin
          call CUTEDGE( $w$ ) ;
          if  $\text{CONTAIN}(w, \Delta(w)) = \text{DEPTH}(w)$  then
            begin
               $\Delta(v) \leftarrow \Delta(v) \cup \Delta(w) \cup \{(v, w)\}$  ;
              if  $\text{UNCONTAIN}(w, \Delta(w)) < \text{CONTAIN}(v, \Delta(v))$  then
                 $\text{CONTAIN}(v, \Delta(v)) \leftarrow \text{UNCONTAIN}(w, \Delta(w))$  ;
            end
          else
            begin
               $\Delta(v) \leftarrow \Delta(v) \cup \Delta(w)$  ;
              if  $\text{CONTAIN}(w, \Delta(w)) < \text{CONTAIN}(v, \Delta(v))$  then
                 $\text{CONTAIN}(v, \Delta(v)) \leftarrow \text{CONTAIN}(w, \Delta(w))$  ;
              if  $\text{UNCONTAIN}(w, \Delta(w)) < \text{UNCONTAIN}(v, \Delta(v))$  then
                 $\text{UNCONTAIN}(v, \Delta(v)) \leftarrow \text{UNCONTAIN}(w, \Delta(w))$  ;
            end
          end
        end
      end
    end
  return ;
end
end

```

図 4-4 [手続き 2 : G, B から Δ を求めるアルゴリズム]

モジュール間のコール関係 $G = (V, A)$ とエフェクト関係 E が与えられたとき、影響範囲を子孫、親、兄弟のモジュールに局限するようにプログラムモジュール群を合併する問題を考察し、 G をとくに有向木に限った場合、合併すべきモジュールの集合のすべてを $|V| + |E| \times G(|V|)$ のオーダーの手数で求めらることを示した。

G が一般のグラフの場合は、現在のところ能率のよいアルゴリズムが見つかっていない。能率のよいアルゴリズムが存在しないという否定的な結果の可能性も含め、検討することが今後にのこされた問題である。

結 論

入出力装置を効率よく使用するデバイスシェアリングシステムの作製ならびに効率のよいプログラムを能率よく作製するためのプログラミング技法の研究について述べた。

本論文で得られた新しい結果および今後の研究課題に関する詳細はすでに各章の結言で示したので、ここでは、各章で得られた成果と今後に残された問題について簡単に述べる。

第1章では、高速のインタフェースプロセッサを介して接続されたミニコンピュータ PDP-11/20 2セットに対し、開発したデバイスシェアリングシステムについて述べた。このシステムにより、コンソール、タイプライタとテレタイプライタを除く全ての入出力装置が効率良く、あたかも自システムの装置であるかのように使用することができるようになり、ラインプリンタなどの装置の使用効率が高まった。

第2章、第3章では、プログラムの長さを短縮するプログラミング技法に関して述べた。

第2章では、ヤーノフのプログラム形が与えられたとき、それと強等価であり、かつ演算節点と分岐節点の節点数の総和が最小であるヤーノフのプログラム形を求める問題について考察し、この問題がいわゆる“決定表”の問題を含んでいることを示した。

第3章では、ヤーノフのプログラム形が与えられたとき、それと動作等価で、多入力サブルーチンを許したプログラム形のなかで、演算節点と分岐節点の個数の和が最小という意味での最簡形を求める問題を考察し、与えられたヤーノフのプログラム形の節点数を $|V|$ とすると

き、 $|V| \log |V|$ に比例する手数で最簡形を求める手続きを示した。

等価な部分をまとめてプログラムの長さを短かくする他の方法としては、フラグを用いてプログラムをまとめる方法がある。このプログラミング技法は任意の数のフラグを許した場合、多入力カフルーシンよりも多くの部分をまとめることができる。このプログラミング技法を用いて等価な部分をまとめ、与えられたプログラム形と等価でかつ節点数の総和が最小であるプログラム形を求める能率のよい手続きは、フラグの数を任意に許した場合、固定した場合のいずれの場合についても、知られていない。この最簡形を求める能率のよい手続きを考察することが今後に残された一つの問題である。

第4章では、サフルーシンなどのモジュール間のコール関係と影響関係が与えられたとき、できるだけそのモジュール構成を保ち（具体的には、できたあとのモジュール数が最大となるように）、影響範囲を子孫、親、兄弟のモジュールに局限するように、プログラムモジュール群を合併する問題を考察し、与えられたモジュール集合 V とコール関係からなる有向グラフを有向木に限った場合、合併すべしモジュールの集合のすべてを $|V| + |E| \times G(|V|)$ に比例する手数で求める手続きを示した。ただし、 E は影響関係を表わすモジュール対の集合。

この手続きを実際に使用する場合、このコール関係 G と影響関係 E をプログラムモジュール群 $\mathcal{M}$ にどのように記述するか問題となるが、この記述の問題は上記の手続きを使用するためだけでなく、プログラムモジュール群のドキュメントの立場からも重要な問題である。

第1章で述べたデハイスシェプリンカシステムは現在のDOSを改造することによって作製されたが、内部は

理を知る方法はリスティングを解説する以外になかった。
リスティングには、モジュール間のコール関係、影響関係がモジュールのための命令群の前に少しだけ記述されていたが、DOS中でのコントロールの流れ（たとえばコマンドが入力されたときに実行されるモジュールの系列）を知るには、それだけでは不十分なものであった。モジュールの内部を解説しなくても全体の構成がわかるように、モジュールの命令群の前にモジュール間のコール関係および影響関係を記述する必要があった。この記述様式に關してはいくつかの提案があるが、⁽¹⁴⁾これらの記述様式に對して判定基準を定め、比較検討することは今後に残された問題である。

謝 辞

本研究の全過程を通じて、直接理解ある御指導を賜わり、つねに励ましていただいた嵩忠雄教授に心から深謝します。

本研究、とくに第1章に関して終始、適切な御指導と御助言をいただいた都倉信樹助教授、藤井護講師、故細見輝政博士に心から感謝します。また、本研究の第2章、第3章、第4章に関して終始、適切な御指導と御助言をいただいた谷口健一博士に心から感謝します。

大学院修士および博士課程において御教示、御指導いただいた情報工学科田中幸吉教授、木沢誠教授、藤沢俊男教授、的場進助教授、志村正道助教授、豊田順一助教授、制御工学科桜井良文教授、坂和愛幸教授、辻三郎教授、白江公輔教授、須田信英教授に心から感謝します。

また、種々の面で御助言、御鞭撻いただいた吉岡信夫博士、電子技術総合研究所鳥居宏次博士に厚くお礼を申し上げます。

さらに、筆者の在学中、御討論いただいた嵩研究室の諸氏に心から感謝します。とくに第1章に関して詳細設計ならびにプログラム作製をしていただいた本田直人氏、第4章において御討論いただいた齊藤孝文氏に心から感謝します。

- (1) ミニコンのソフトウェアとネットワーク報告集, 情報処理学会プログラミング・シンポジウム (1972-07).
- (2) PDP-11 handbook; DEC (1970).
- (3) DA 11-D INTERPROCESSOR COMMUNICATIONS CHANNEL, DEC (FEB. 1972).
- (4) PDP-11 Disk operating system monitor programmer's handbook, DEC (1971).
- (5) PDP-11 Disk operating system functional specification, DEC (1970).
- (6) J.D. Rutledge: "On Ianov's program schemata", JACM, 11, 1, p.1 (Jan. 1964).
- (7) Z. Manna: "Program schemas", in A.V. Aho (ed.): "Currents in the theory of computing", Prentice-Hall, Inc. (1973).
- (8) J. Hopcroft: "An $n \log n$ algorithm for minimizing states in a finite automaton", in Z. Kohavi and A. Paz (eds.): "Theory of machines and computation", Academic Press (1971).
- (9) L.I. Press: "Conversion of decision tables to computer programs", Comm. ACM, 8, 6, p.385 (June 1965).
- (10) R. Tarjan: "Finding dominators in directed graphs", SIAM J. Comput., 3, 1, p.62 (Mar. 1974).
- (11) W.P. Stevens, G.J. Myers and L.L. Constantine: "Structured design", IBM Syst. J., 13, 2, p.115 (1974).
- (12) A.V. Aho, J.E. Hopcroft and J.D. Ullman: "The design and analysis of computer algorithms", Addison-Wesley Publishing Company (1974).
- (13) A.V. Aho, et al.: "On finding lowest common ancestors in trees", ACM Sympo. on Theory of Computing (April 1973).
- (14) D.L. Parnas: "A technique for software module specification with examples", Comm. ACM, 15, 5, p.330 (May 1972).

