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

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，当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も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，多入口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，できるだけもと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
デバ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―― 9
デバイスシェアリングシステム

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　デバイスのロック・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バイス ―― 22

1.5.3　ファイル構造を持つデバイス ―― 24

1.6　デバイス管理ルーチン（DEMON） ―― 27

1.6.1　DEMONの機能 ―― 27

1.7　結言 ―― 35

第2章　ヤーノ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　サブルーチン化によるプログラム形の簡単化 -- 49
3.1　序言 ---------- 49
3.2　定義と問題 ---------- 51
3.3　サブルーチン可能グラフ ---------- 58
3.4　サブルーチン化の条件 ---------- 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　$\bar{S}$の各節点元のサブエンドを求める。---- 61
3.5.3　$\bar{S}$の節点の分割π($\bar{S}$)を求める ---------- 62
3.5.4　$\tilde{S}$の定義 ---------- 64
3.6　$\tilde{S}$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と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ことに関する研究。［Ⅲ］ヤーノフのプログラム形が与えられたとき，多入口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ことに関する研究。［Ⅳ］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，できるだけもと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Ⅰ章には，高速インタフェイ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，現在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，動作等価なものの内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ヤーノフ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，サブルーチンを用いて，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)として公表したサブルーチン化によるプログラム形の簡単化について述べている。

多入口サブルーチンを許したプログラム形とは，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用い，サブルーチ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先頭に書かれた

番地へ戻る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口サブルーチンを許したプログラム形においても，以下のように拡張して使用する。多入口サブルーチンを許した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）の多入口サブ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を終点とするサブルーチン可能グラフとは，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を終点とするサブルーチン可能グラフを求める。つぎに，すべてのサブルーチン可能グラフ内で動作等価なものが一つであればかならずまとめるようにする。

なお，スタック節点，リターン節点を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ブルーチンとか小さなサブルーチ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。

サブルーチ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，

デバッグ，改良などの立場からも重要なことである．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A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ぎつぎとコールされてゆく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程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，できるだけもと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）のオーダの手数で求められることを示した．ただし，$G(n)$ は Hopcroft ら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，ミニ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：“ヤーノ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：“サブルーチ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．それにとも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とこれらを結ぶ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）のもと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．そこで，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ニ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

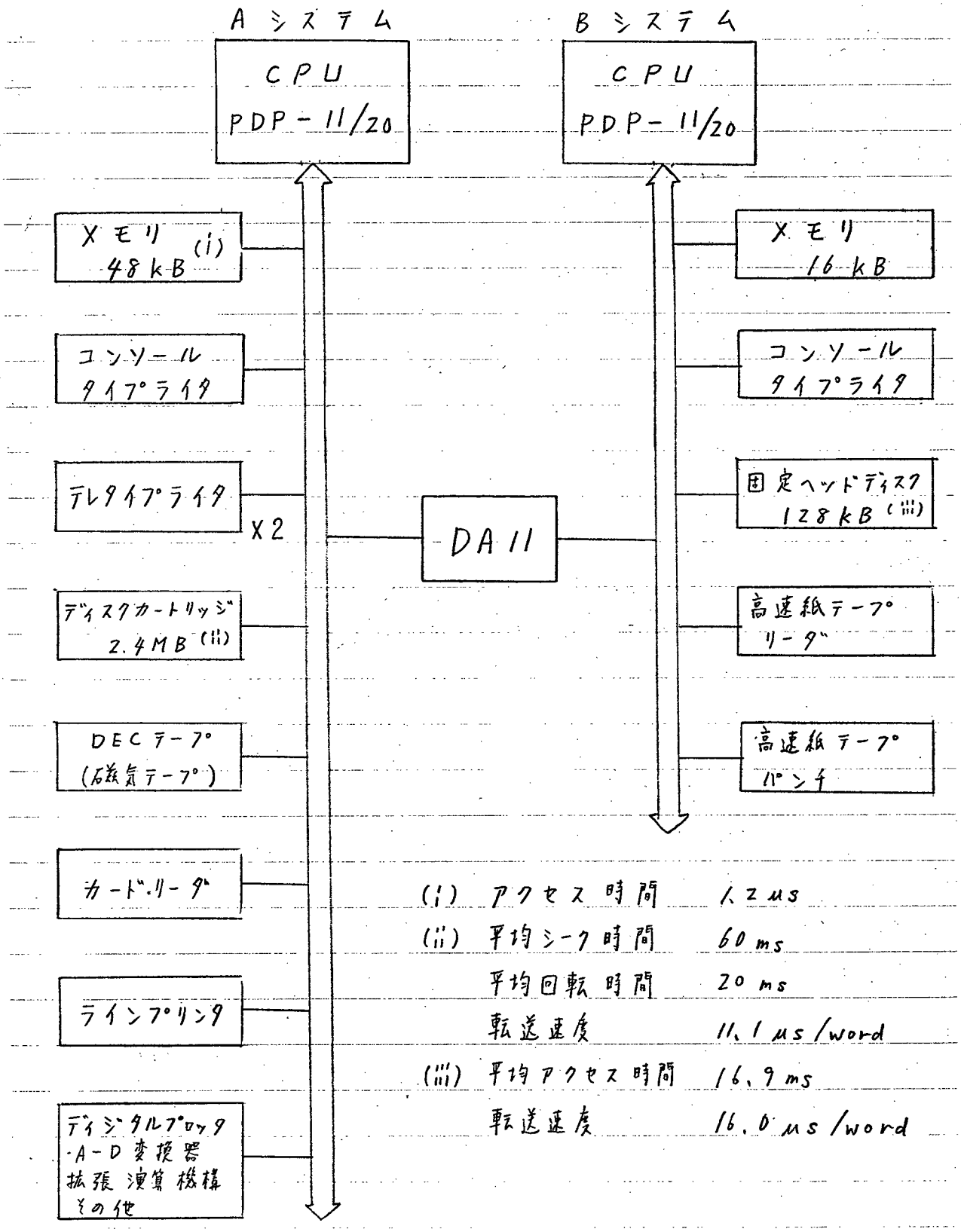


図 1-1 デュアルシステムの構成

が，既存のＤＯＳ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ＤＯＳ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，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ＯＰＥＮ要求によりデバイスを占有し，ＣＬＯＳＥ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ＤＥＣテープ）に対しては，ユーザプログラムからのＯＰＥＮ要求によっ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ＲＥＡＤ・ＷＲＩＴＥ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ＰＤＰ－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，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できる方法である。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。

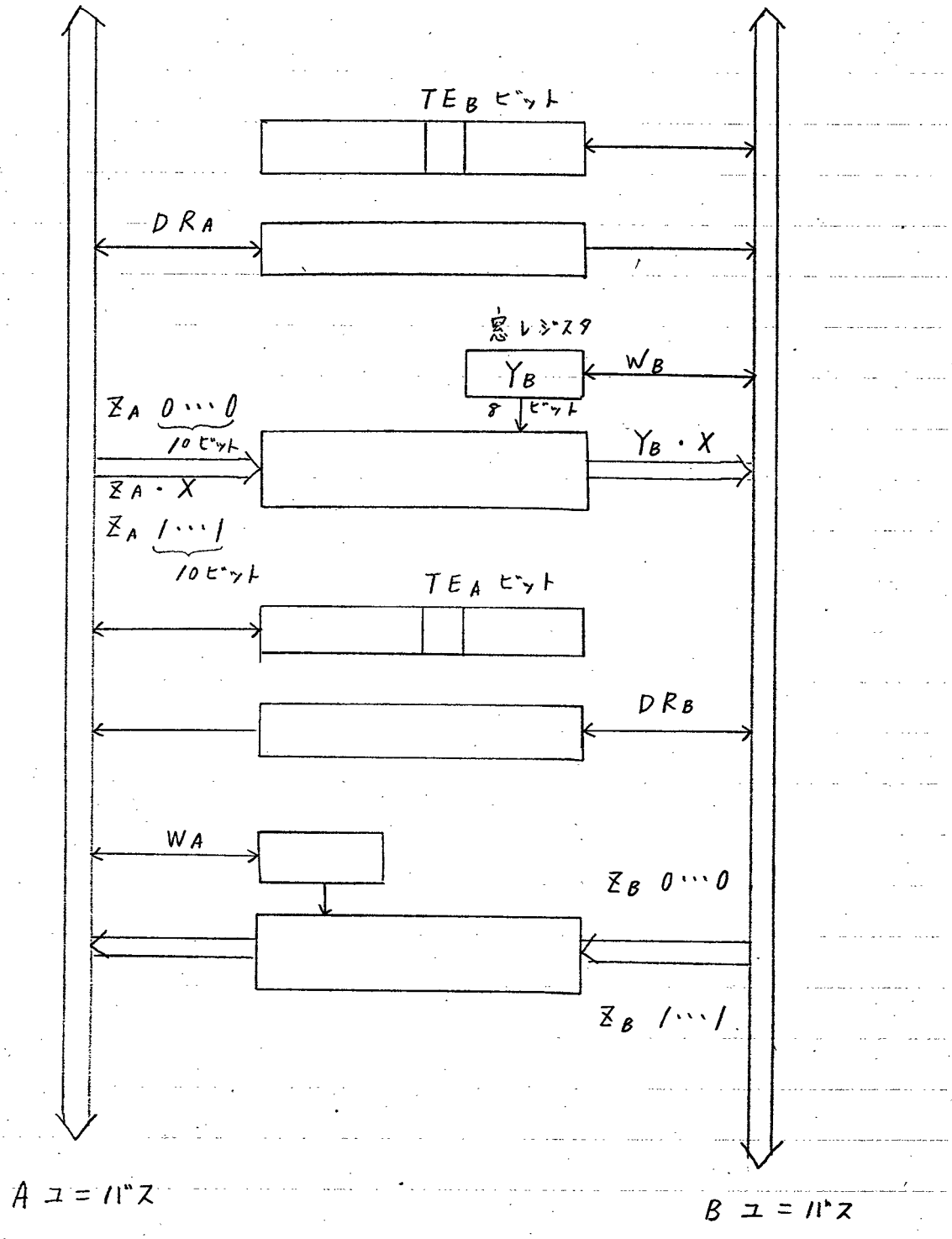

TE B ビット
DR A
退レジスタ
YB
WB
8 ビット
ΣA 0…0
10 ビット
ΣA・X
YB・X
ΣA 1…1
10 ビット
TE A ビット
DR B
WA
ΣB 0…0
ΣB 1…1
A ユニバス
B ユニバス

図 1-2　　D A 11

ドレス $Y_B X$ へあたかも自分の側のアドレススペースであるが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。

DA11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 の内容できまる。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DA11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もっている。

PDP-11には千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 DA11を経由してBの $Y_B \cdot X$ の番地以降へCPUA，Bをわずらわせずに転送できる。

(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DA11中のデータレジスタ DR_A（2バイト）にデータを送ると，DA11はB側に割込みをかける。B側は，DR_A 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 DOSモニタ

○DOS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）

○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，LIO）：データのブロッキング，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を行う。

(ii) フィジカルI/O[+]（略して，PIO）：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。

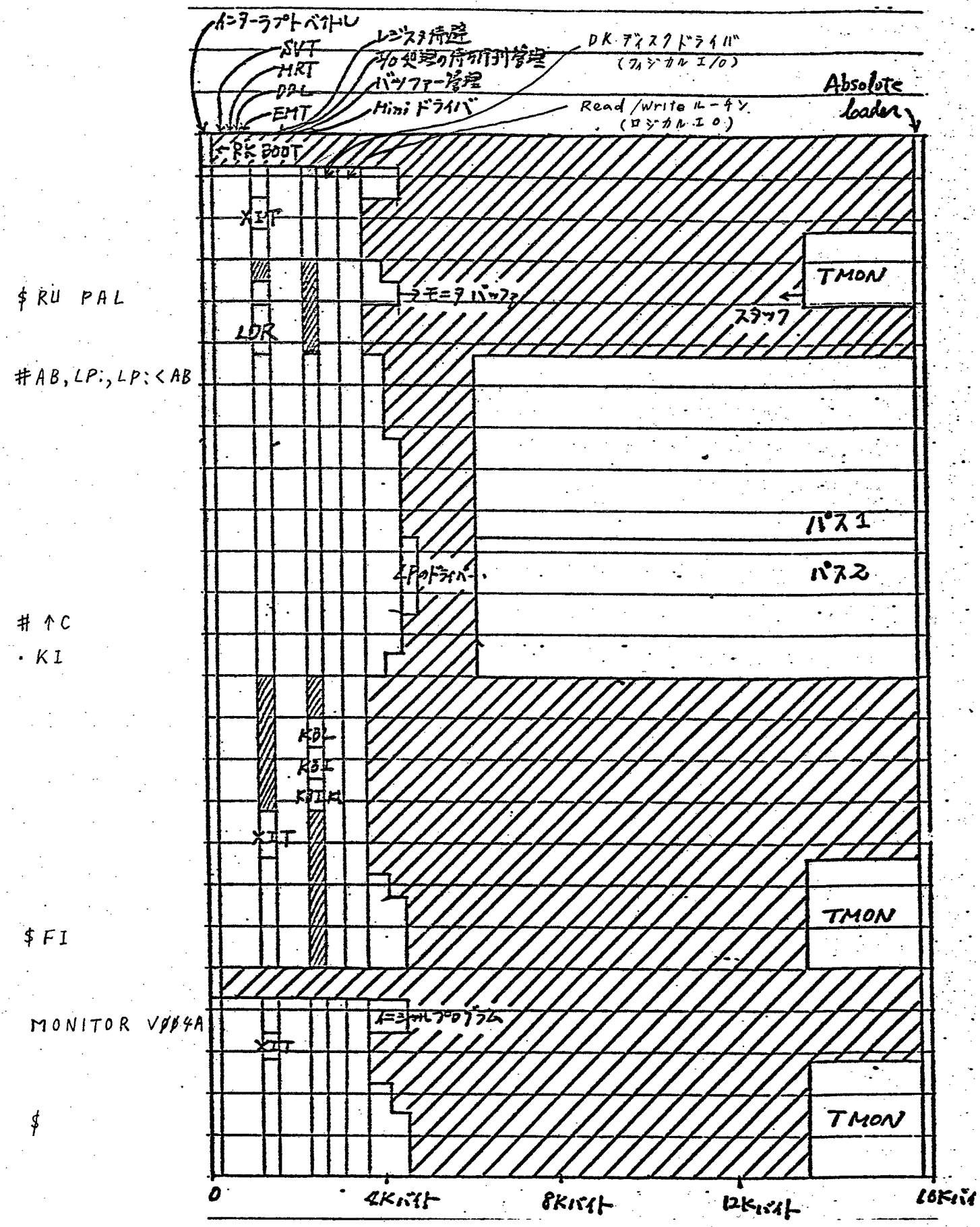

インターラプトベクトル
SVT
MRT
DRL
EMT
レジスタ待避
I/O処理の待ち制御管理
バッファー管理
Miniドライバ
DK ディスクドライバ
(フィジカル I/O)
Read/Write ルーチン
(ロジカル I/O)
Absolote loader
←RK BOOT
XIT
TMON
ラモニタバッファ
スタック
LDR
$RU PAL
#AB,LP:,LP:<AB
パス1
パス2
LPのドライバー
#↑C
・KI
KBL
KBL
KBIN
XIT
TMON
$FI
MONITOR V004A
イニシャルプログラム
XIT
TMON
$
0
4Kバイト
8Kバイト
12Kバイト
16Kバイ

図 1-3 コアマップ

§1.3　システム作製の基本方針

前節1.2.2でも述べたように，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応じ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も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．

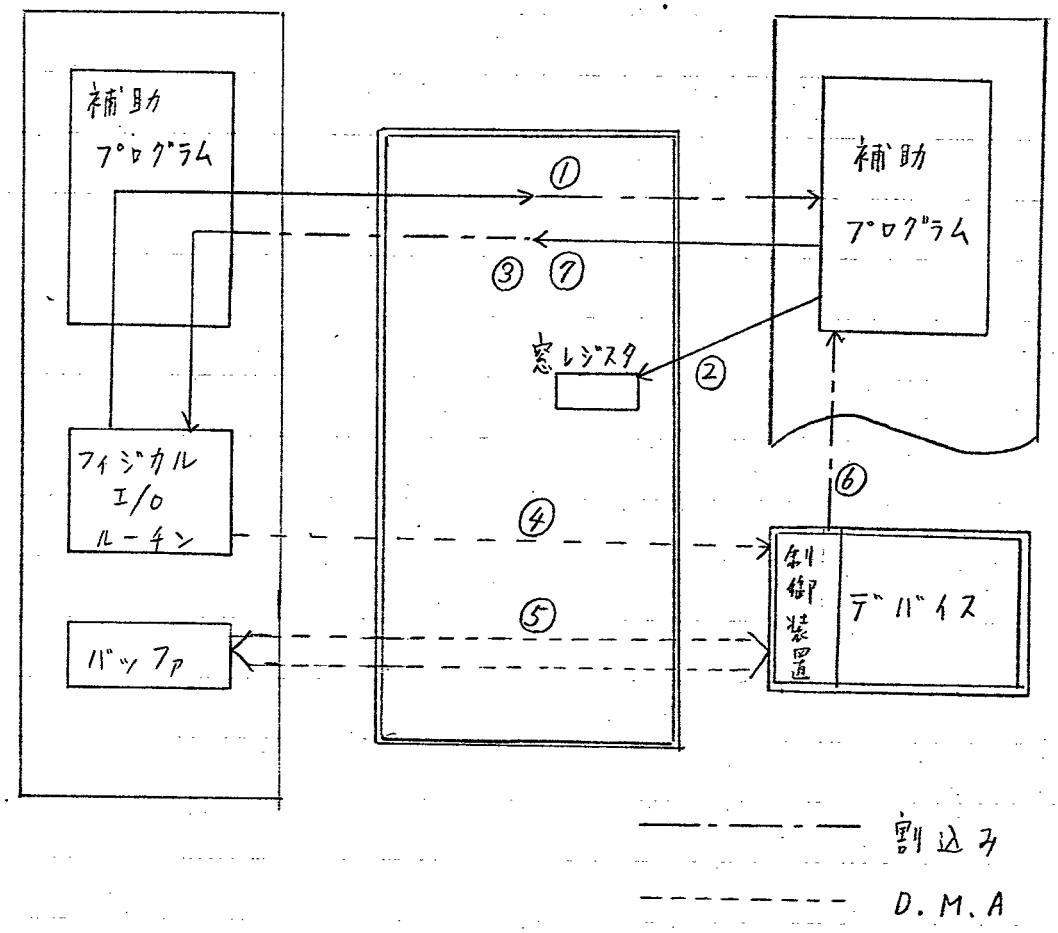
(iv) デバイスの占有時間をできるだけ短くすること．

§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PIOが直接（デバイスが相手側であれば，DA11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CPU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DA11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，従来どおり上の④⑤⑥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 窓レジスタの設定依頼（P.T.方式）
② 窓レジスタの設定
③ 窓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 デバイスからの割込み
⑦ 割込み伝達（P.T.方式）

図 1-4 相手システム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（モニ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，各デバイスについて，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，ラインプリンク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も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いとすれ

ば，デバイスa，bは占有ロックが解除されることは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あ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．引き続く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

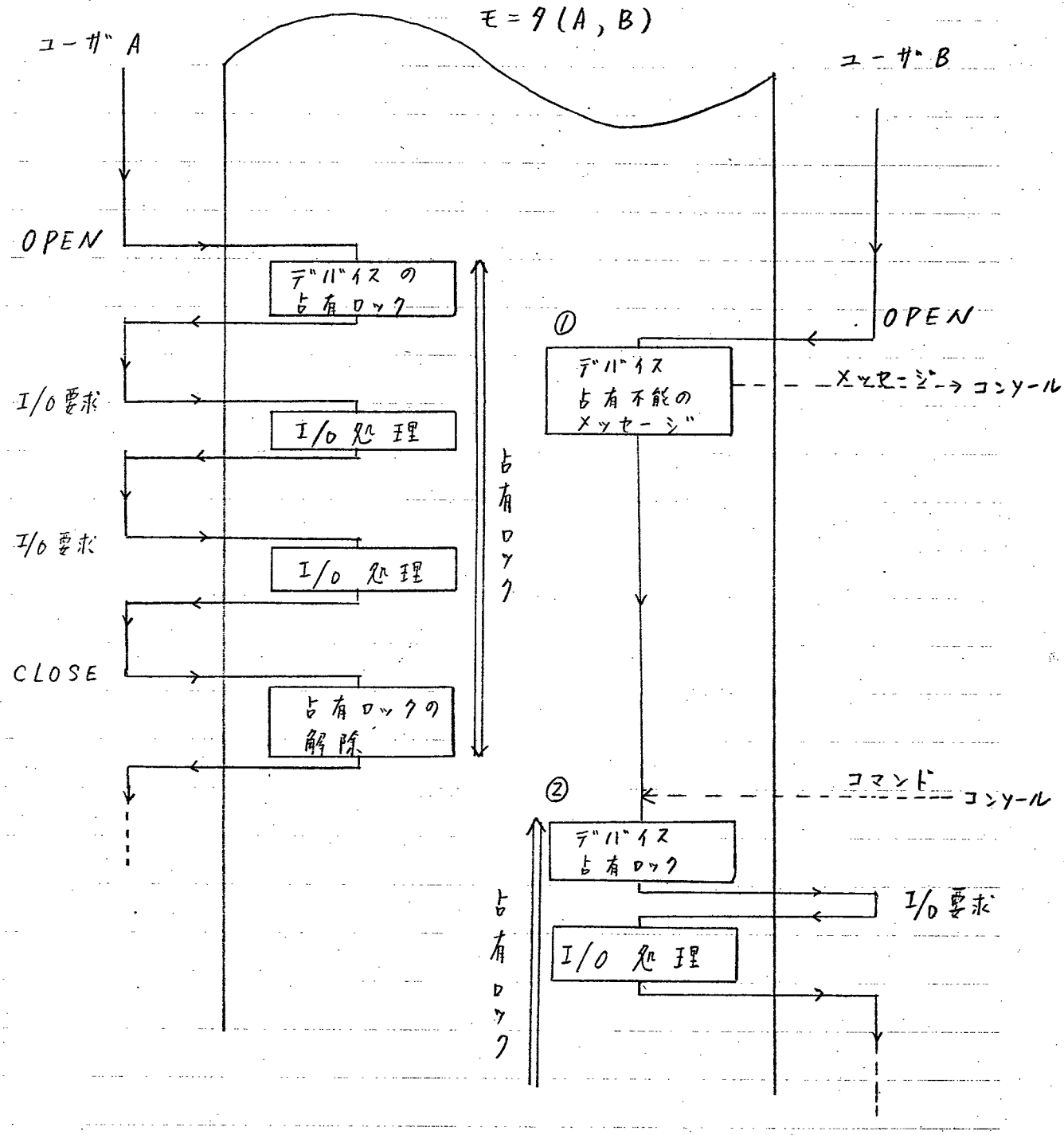


図 1-5　ラインプリンタやカードリーダのロック・アンロック

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ニタを一つにまとめて書く。）②では，OPENマクロ受付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力し，オペレータの指示を待っている。②でオペレータが，そのデバイスがあい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，これにより複数個のファイルを管理している。ファイルは複数個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。

入出力とも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力するための）マクロ，あるいはOPENE（すでに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，モニタは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。引き続く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の特定の場所に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に返す際に行われる。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ノ-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。

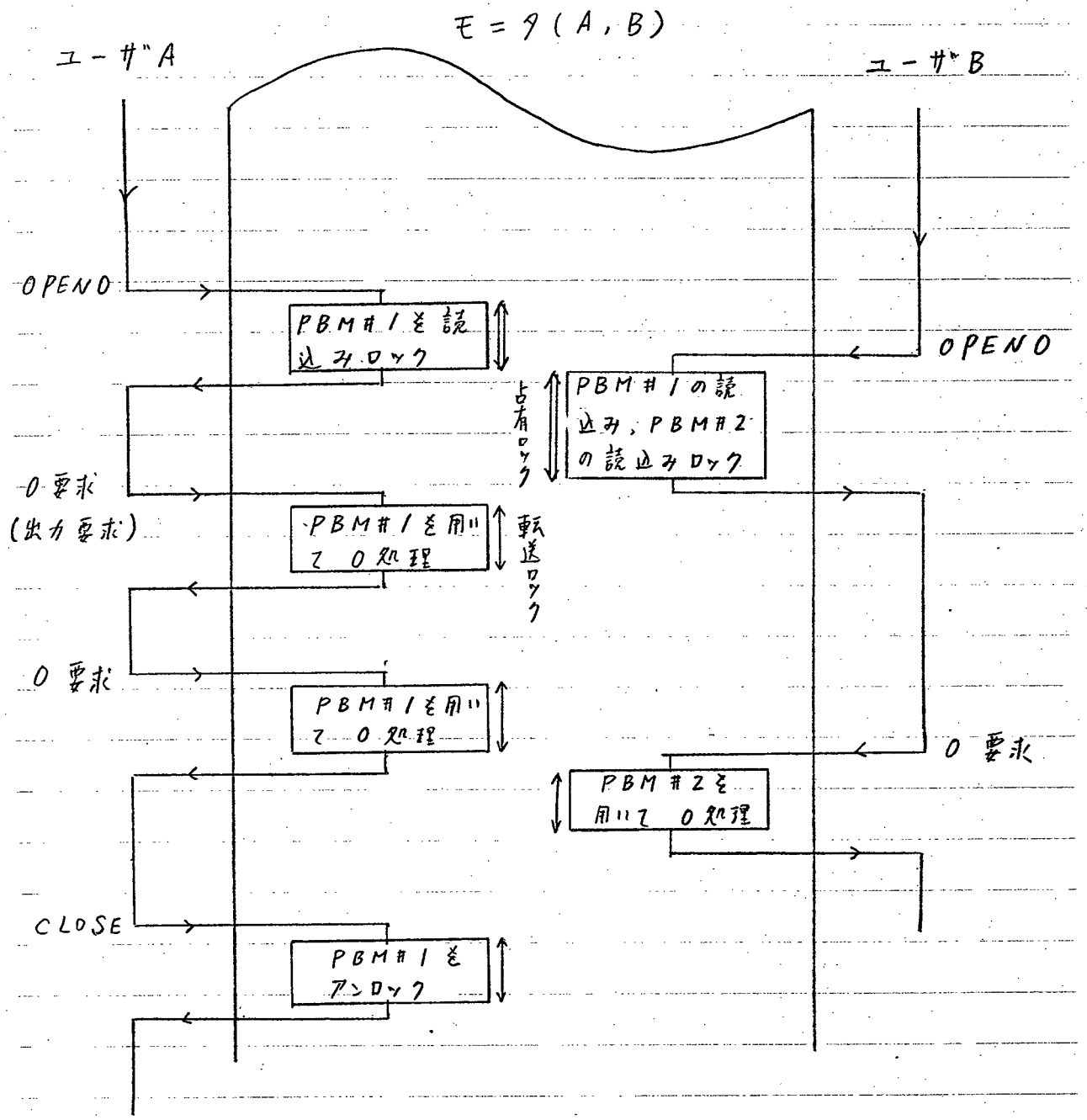


図 1-6 ディスクのロック・アンロック

§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DA11は一種のリソースとしてDEMONが管理する

DEMONは両システムのモニ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

(i)　デバイス管理テーブル

1.5節でも述べたように，モニタ中の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ればここで調べ，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3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（TT_A：TT_{AA}およびTT_{AB}）

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（OT_A：OT_{AA} およびOT_{AB}）

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。

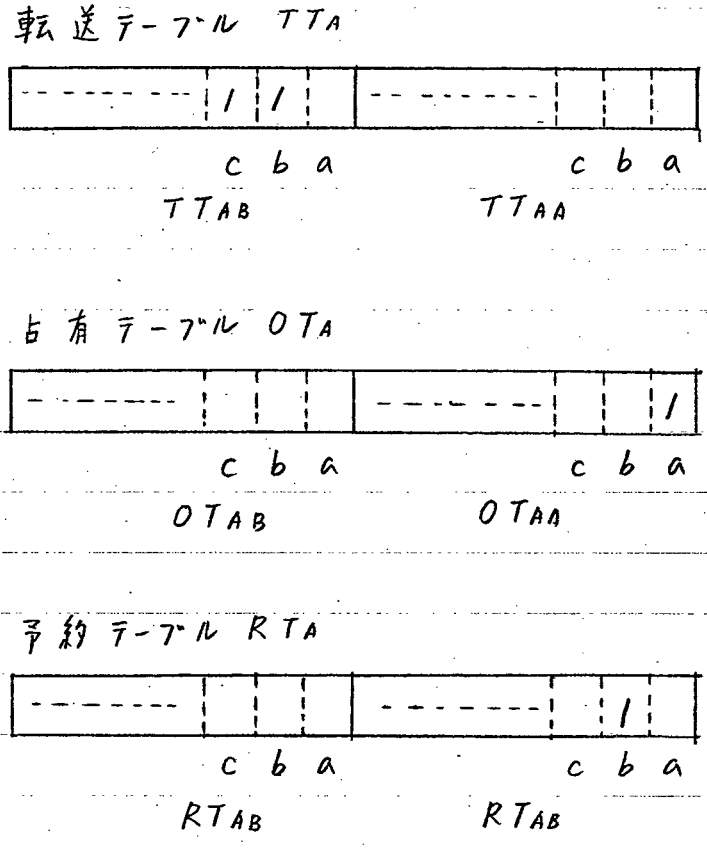


図1-7　デバイス管理テーブル

○予約テーブル（RT_A；RT_{AA} および RT_{AB}，ST_A）

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 RT_A 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 ST_A 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6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，RT_{AA} の対応するビットがセットされている。

(ii) 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DEMON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り口であり，⬭はB側のDEMONからの割込みに対する処理の入り口である。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

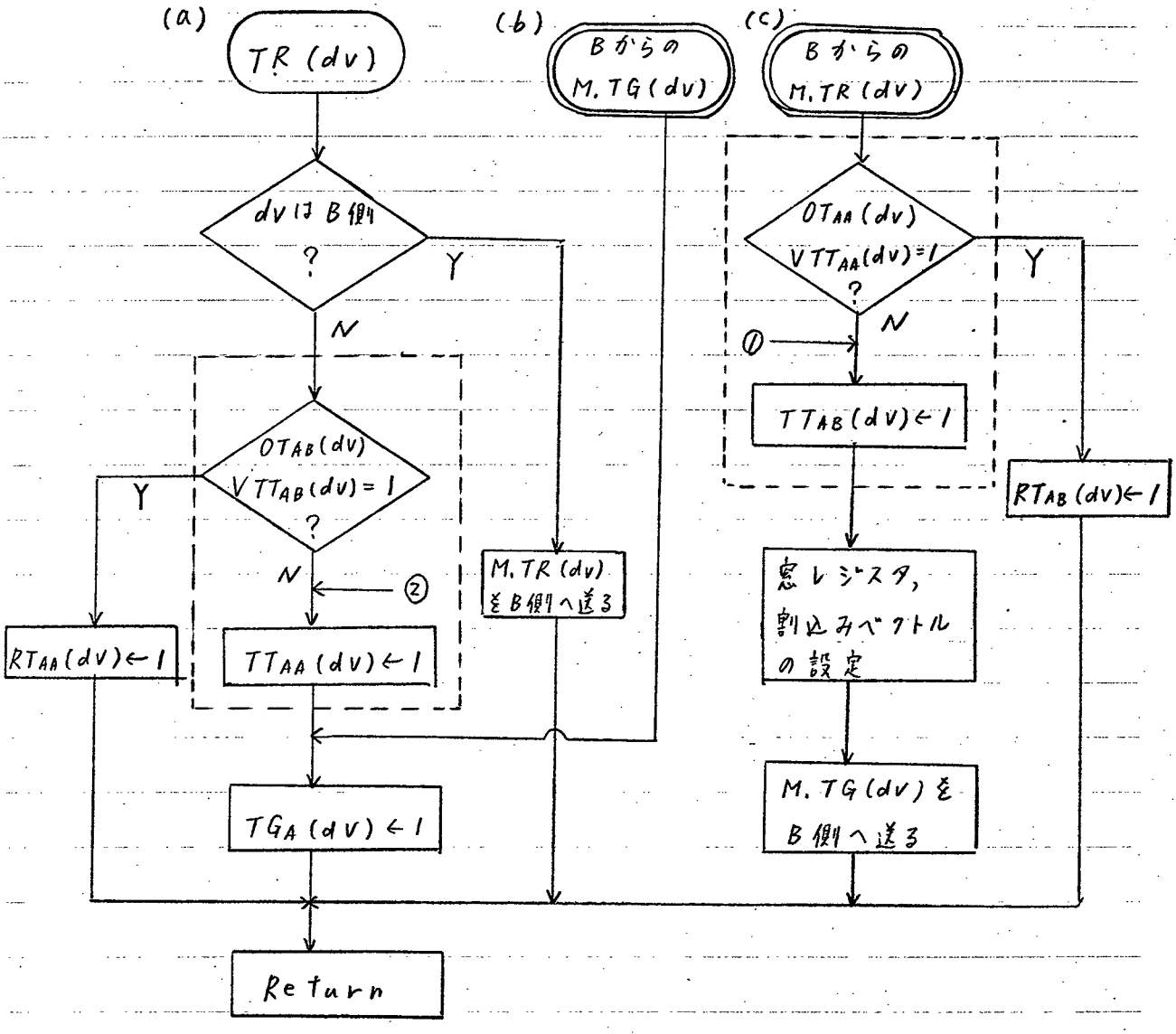


図 1-8 A側 DEMON の転送ロック要求の処理

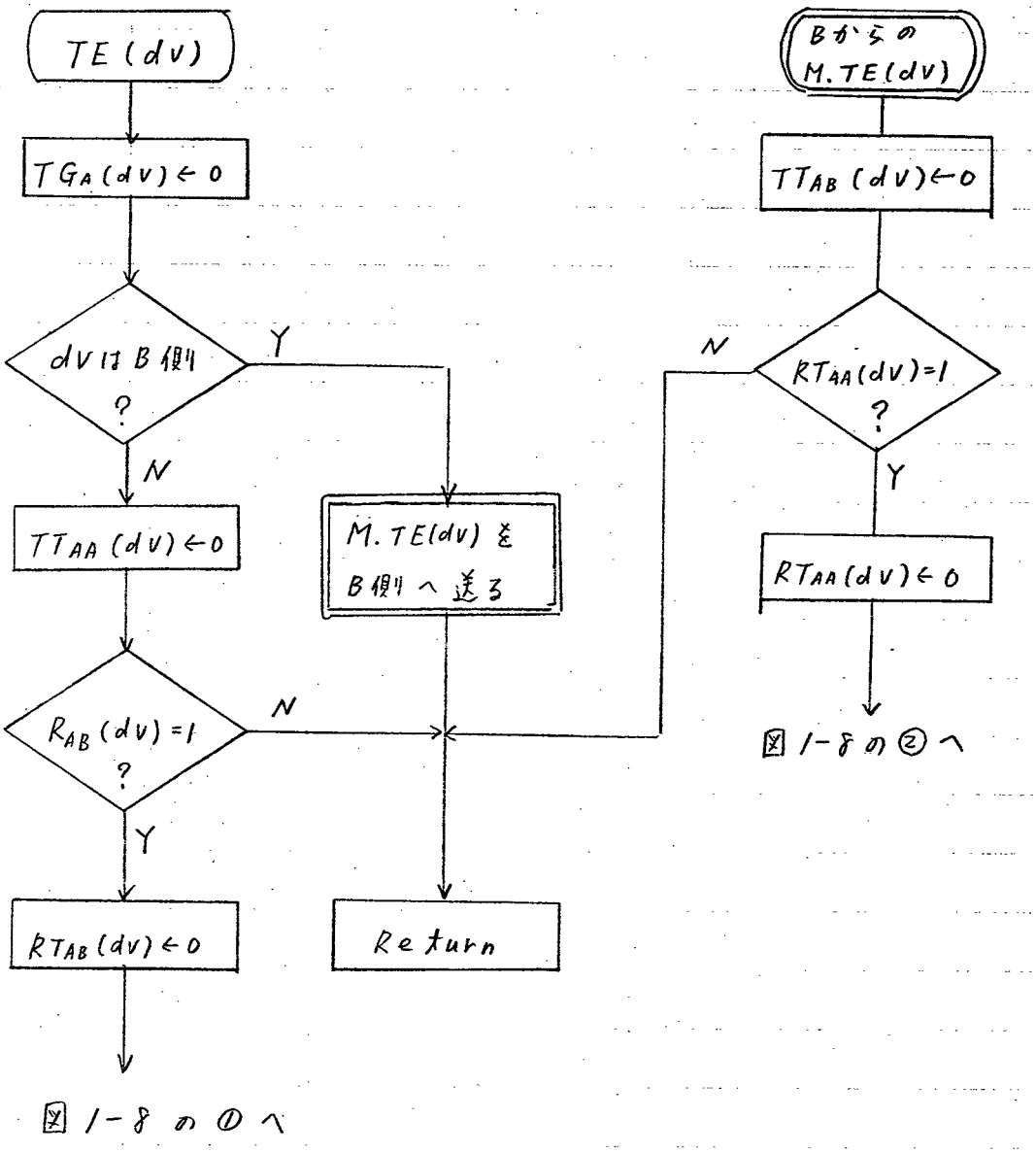


図 1-9　A側 DEMON の転送ロック解除 の処理

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

PIOからデバイス名(1)をパラメタとする転送ロック要求TR(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$TT_{AA}(1)$および PIOとの通信領域である$TG_A(1)$をセットし，転送ロック許可を知らせる．PIOは，コントロールが戻ると常に$TG_A(1)$を見ており，$TG_A(1)$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DEMON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DEMONに処理を依頼する(M.TR(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$RT_{AB}(1)$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PIOに戻るが，B側の通信領域$TG_B(1)$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$TG_A(1)$をクリアし，A側のデバイスであるから$TT_{AA}(1)$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$RT_{AB}(1)$をセットされているため，図1-8の②へとび，転送テーブル$TT_{AB}(1)$をセットし，DA11の窓レジスタに，Bシステムから DA11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な値を設定する。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，$TT_{AB}(1)$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$RT_{AA}(1)$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)が用意されていないことは，前にも述べたとおりである。

(v)　その他のDEMON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DEMON中の割込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DEMON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。

DEMON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．

§八.7　結言

デバイス管理ルーチンの作成およびフィジカルIOルーチン，ロジカルIO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/0000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，モニタの常駐部の増加分は約/000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を解読したが，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/9/2年/2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ェアリングシステムを作製するためミニコンピュータPDP-11/20のディスク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ルとただ一つのレジスタX（ただし，無限の情報がはいりうる）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J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 $\langle S, \mathcal{I} \rangle$ になる。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 $\mathcal{I}$ の下で等価であるとき，S_1, S_2 は弱等価であるという。

この章では，任意のヤーノフのプログラム形 S が与えられたとき，それと弱等価で，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ノフのプログラム形 S_m を見つける問題を考察する。ヤーノ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。準プログラム形の簡単化は，完全に指定された決定性有限オートマトンの簡単化の問題と同じである[6]。この章では，求めるヤーノ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，もとのヤーノフのプログラム形にもどすとき，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。

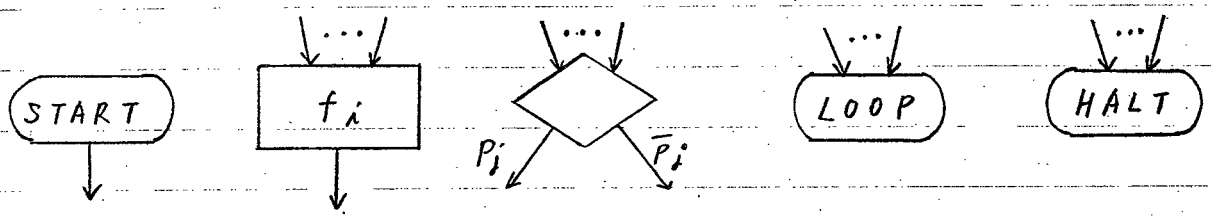
§2.2 定義と問題

（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 $\overline{P_j}$ をラベルとしてもつ。

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 $\overline{P_j}$ の2本の出力枝をもつ分岐節点では 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 $\overline{P_j}$ 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力して停止する。

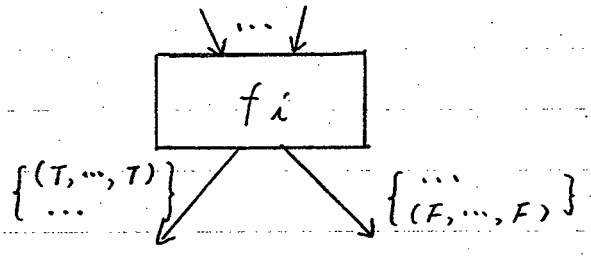
プログラム形 S_1，S_2 が解釈 $\mathcal{I}$ の下で等価であるとはプログラム $\langle S_1, \mathcal{I} \rangle$，$\langle S_2, \mathcal{I} \rangle$ が各入力初期値について，一方が終了節点に達して停止すれば他の方もそうであり，かつ，そのときの出力変数の内容（計算の最終結果）が等しいことをいう。S_1，S_2 が任意の解釈 $\mathcal{I}$ について等価であるとき，強等価であるといい，$S_1 \equiv S_2$ と書く。

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　ヤーノフ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$P_1, \cdots P_m$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）．$P_1, \cdots, P_m$による判定結果が$a_1, \cdots, a_m$（各a_iはTかF）であれば，$(a_1, \cdots, a_m)$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，$\{T, F\}^m$の任意の元はそ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をもち，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 ρ 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 $\rho(\bar{v})$ 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 ($\subset \{T,F\}^m$) とする。一方、S_{min} の節点 $\rho(\bar{v}_1)$ から $\rho(\bar{v}_2)$ 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 $P_1, \cdots, P_m$ による判定結果 $(a_1, \cdots, a_m)$ (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。

§2.4 定理の証明

S_{min} の任意の節点 v に対し，S_{min} の開始節点から 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 $\mathcal{I}$，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 $C(\bar{S}_{min}, \mathcal{I}, \xi)$ と書き，そのk番目（開始節点を0番とする）の節点を $C_k(\bar{S}_{min}, \mathcal{I}, \xi)$ で表す。$C(S_{min}, \mathcal{I}, \xi)$ 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 $C_k(S_{min}, \mathcal{I}, \xi)$ で表す。

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

$C_k(\bar{S}_{min}, \mathcal{I}, \xi) = \bar{v}$ となるようなすべての $\mathcal{I}$, ξ, k について，節点 $C_k(S_{min}, \mathcal{I}, \xi)$ の集合

と定義する。$\sigma(\bar{v})$ 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，$v \in \sigma(\bar{v})$ とすると，$\bar{S}_{min}(\bar{v}) \equiv S_{min}(v)$ である。

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

更に $\bar{S}_{min}(\bar{v}) \neq S_{min}(v)$ を示すような解釈 $\mathcal{I}'$, 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 $\mathcal{I}^H$, 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 $\mathcal{I}^H$ を作るため, まず, S_{min} のヘルブランド領域 Hs_{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 に現れる個体定数なら "x_i" と "a_i" は Hs_{min} の元である。f_i が S_{min} に現れる関数記号で "t_j" が Hs_{min} の元なら, "$f_i(t_j)$" は Hs の元である。

ヘルブランドの解釈 $\mathcal{I}^H$ をつぎのように作る。S_{min} に現われる各個体定数 a_i には Hs_{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 Hs_{min} の記号列 "t_j" を Hs_{min} の記号列 "$f_i(t_j)$" に写像するような関数を割当てる。

S_{min} の述語記号と個体定数 ξ', ξ'' への割当に関しては, $\bar{S}_{min}$, S_{min} における $\mathcal{I}^H$, ξ^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, $C(\bar{S}_{min}, \mathcal{I}, \xi)$, $C(S_{min}, \mathcal{I}, \xi)$ に一致し, それ以降はそれぞれ, $C(\bar{S}_{min}(\bar{v}), \mathcal{I}', \xi')$, $C(S_{min}(v), \mathcal{I}', \xi')$ 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] の $(\bar{v})$ は 1 個かただ 1 個の節点を含む。

(証明) もし $v_1, v_2 \in \sigma(\bar{v})$ 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，$\rho(\bar{v}_1)$，$\rho(\bar{v}_2)$ が同じ $v \in V$ を含むことはない。

（証明）もし，$v \in \rho(\bar{v}_1)$，$v \in \rho(\bar{v}_2)$ と仮定すると補題 2-2 より $S_{min}(v) \equiv \bar{S}_{min}(\bar{v}_1) \equiv \bar{S}_{min}(\bar{v}_2)$ がなりたつ。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，ρ は $\bar{V}$ から V への 1 対 1 対応と考えられる。定理の P をこの ρ 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。$v_E \in \rho(\bar{v}_E)$，$v_I \in \rho(\bar{v}_I)$ であることは $S_{min} \equiv \bar{S}_{min}$ であることからあきらか。$v_L \in \rho(\bar{v}_L)$ 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 ρ は条件 (a) を満足する

(ii) 補題 2-2 より ρ 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 $v_2' \in \rho(\bar{v}_2)$ 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，$v_2 \in \rho(\bar{v}_2')$ 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 ρ は条件 (C) を満足する。

§2.5 結言

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，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。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[7]を，複数個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。多入口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，ただヤーノフタイプ（有限オートマトン的）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る．

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．

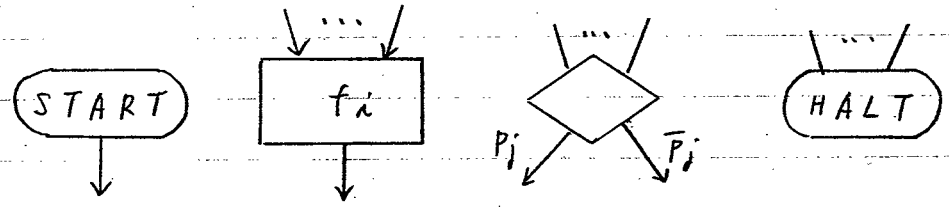
この章では，任意のヤーノフのプログラム形が与えら

れたとき，それと動作等価で，演算節点と分岐節点の個数の和が最小の多入口サブルーチンを許した（すなわち，スタック命令，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。(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 $\overline{P}_j$ をラベルとしてもつ。(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いる。)

このプログラム形 S に現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, 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が示す演算を実行する。ラベル P_j および $\overline{P}_j$ の二本の出力枝をもつ分岐節点では, 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 $\overline{P}_j$ 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。



開始節点　演算節点　分岐節点　終了節点

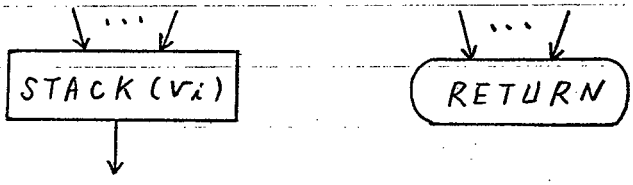
図3-1　ヤーノフのプログラム形の命令

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って終了節点に至るような計算の道（ある解釈，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っ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 $\mathcal{F}$ とする．

つぎに，ヤーノフのプログラム形に，多入口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もどし，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，ヤーノフ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，$F \subseteq R$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っ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$S \in F$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$\overline{P_j}$の記号系列（v_1が開始節点のときは“ＳＴＡＲＴ”という記号を含める）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，$L(v_1, v_2) = \{w \mid v_1 \xrightarrow{w} v_2$，かつ，途中で節点$v_2$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)”や“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$\mathcal{F}$，$\mathcal{R}$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$S\in\mathcal{F}$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

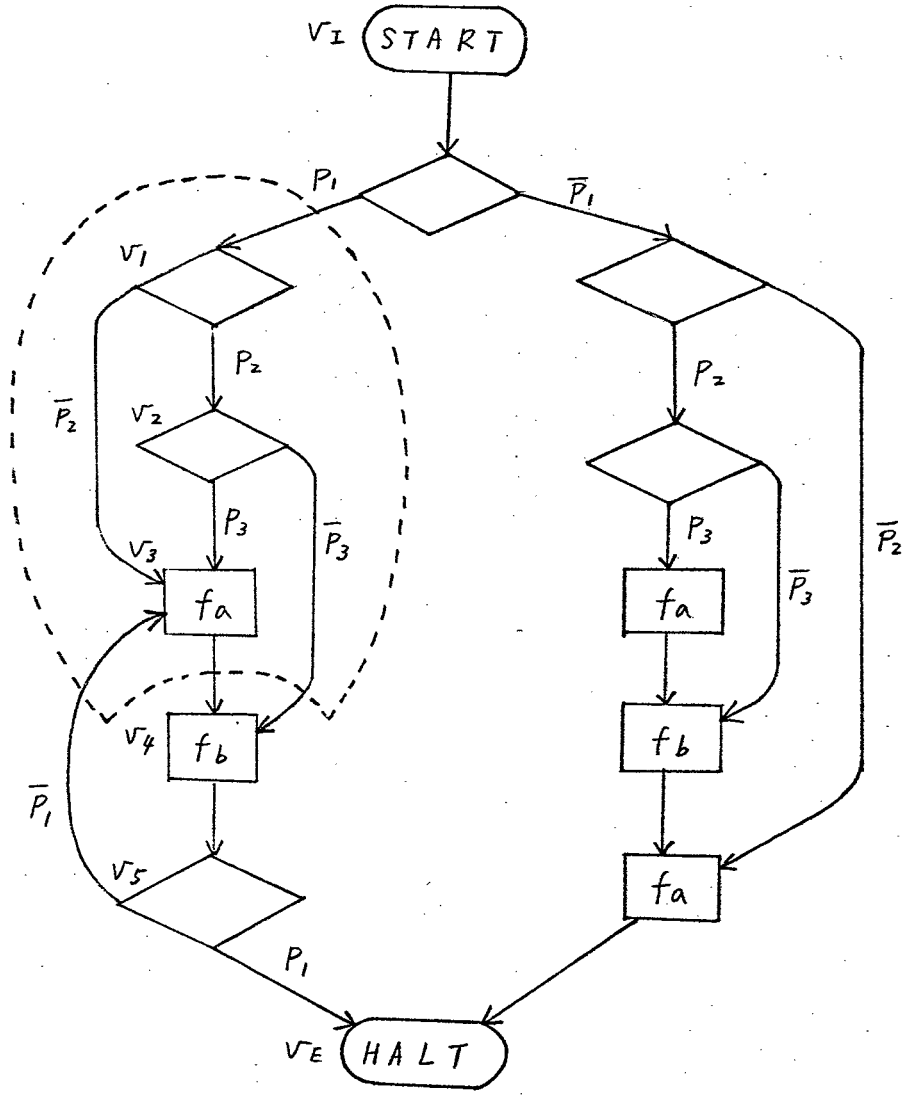


図 3-3　Π_1 に属するプログラム形の例 S_1

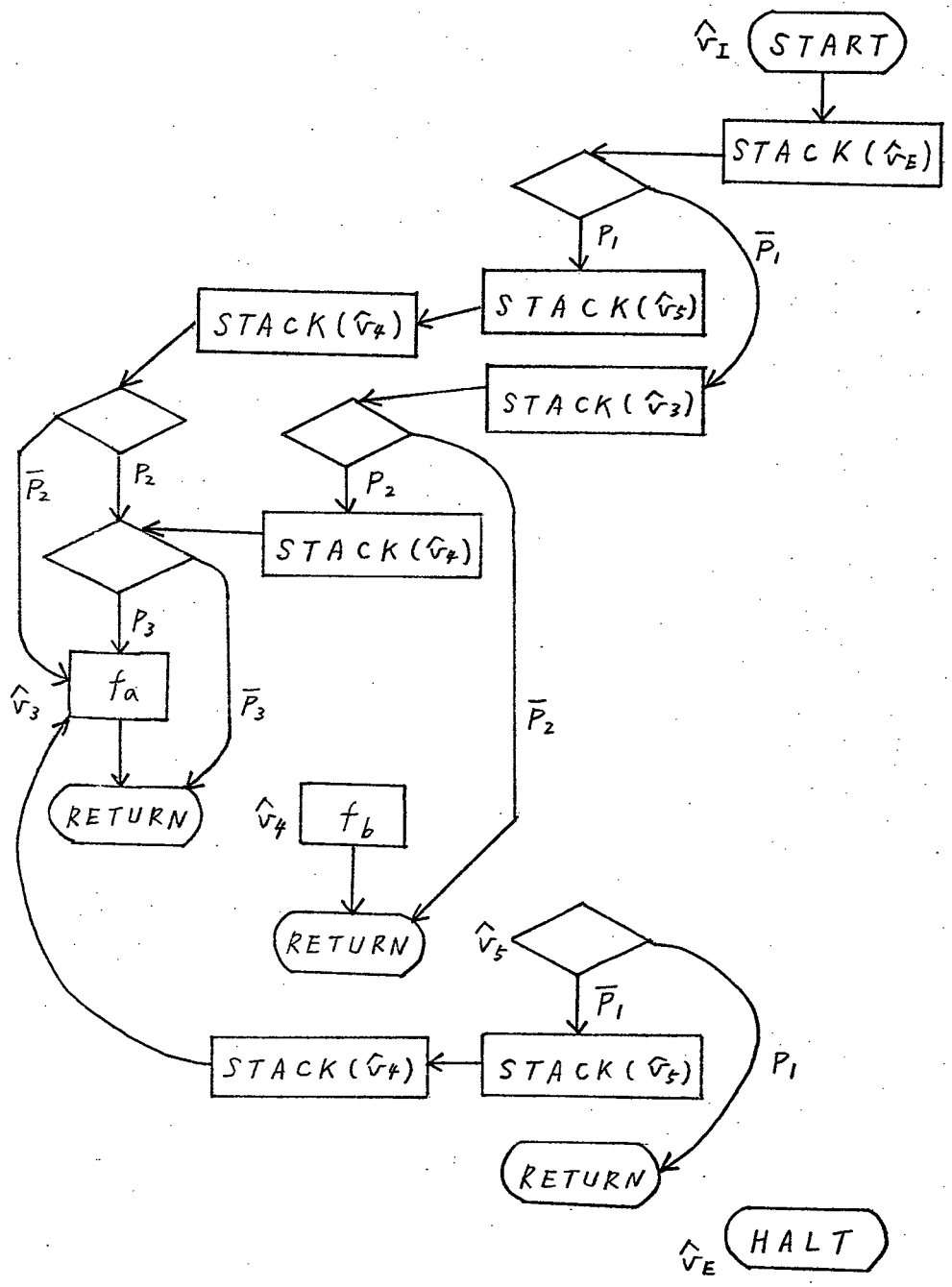


図 3－4　　R に属するプログラム形の例 S_2

は，能率のよいアルゴリズムはないので，以下では，おもに，$C_2 = 0$ 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．

§3.3 サブルーチン可能グラフ

クラス $\mathcal{H}$ のプログラム形に関して，有向グラフと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，s 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 $E^1(v)$（$E(v)$ とも書く．），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，$l(v)$ 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 $E^1(v_2)=v_4$，$E^2(v_1)=v_5$，$E^3(v_1)=v_E$ 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 $\{v' \mid E^k(v')=v,\ k=1,2,\cdots\}$ 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れ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 $\mathcal{R}$ 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〕 $\bar{S}$ の異なるどの2つの節点 $\bar{v}_1$，$\bar{v}_2$ に対しても，$L(\bar{v}_1, \bar{v}_E) = L(\bar{v}_2, \bar{v}_E)$ となることはない．

〔補題3-3〕 $\bar{S}$ において，二つの節点 v_1，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，$L(\bar{v}_1, E(\bar{v}_1)) = L(\bar{v}_2, E(\bar{v}_2))$，$L(E(\bar{v}_1), E^2(\bar{v}_1)) = L(E(\bar{v}_2), E^2(\bar{v}_2))$，…，$L(E^{k_1-1}(\bar{v}_1), E^{k_1}(\bar{v}_1)) = L(E^{k_2-1}(\bar{v}_2), E^{k_2}(\bar{v}_2))$ でなければならない．

（補題3-3の略証） ξ_1，ω'，$\omega'' \in L(\bar{v}_1, E(\bar{v}_1))$，$\bar{v}_2 \xrightarrow{\omega'} \bar{v}_2'$，$\bar{v}_2 \xrightarrow{\omega''} \bar{v}_2''$，$\bar{v}_2' \neq \bar{v}_2''$ 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 に対し，$\bar{S}$ の節点の集合の（v）をつぎのように対応させる．

+ S においても，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，$\hat{S}$ において，$\hat{v}_I \xrightarrow{w} \hat{v}$，$\bar{S}$ において，$\bar{v}_I \xrightarrow{w} \bar{v}$ であるなら，$\bar{v} \in \sigma(\hat{v})$ である．逆に，$\bar{v} \in \sigma(\hat{v})$ 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 $\mathcal{H}$ 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 $\hat{S}$ との関係を述べている．

［補題 3-4］ $\hat{S}$ の節点 $\hat{v}$，$\bar{S}$ の節点 $\bar{v}_i$ に対し，$\bar{v}_i \in \sigma(\hat{v})$ とすると，ある $k_i > 0$ が存在して，$T(\hat{v}) = L(\bar{v}_i, E^{k_i}(\bar{v}_i))$ 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，$\bar{v}_i$，$\bar{v}_j \in \sigma(\hat{v})$ となるような $\hat{S}$ の節点 $\hat{v}$ が存在すれば，$L(\bar{v}_i, E(\bar{v}_i)) = L(\bar{v}_j, E(\bar{v}_j))$ でなければならない．

（証明） $\bar{v}_i$，$\bar{v}_j \in \sigma(\hat{v})$ 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，$L(\bar{v}_i, E(\bar{v}_i)) = L(\bar{v}_j, E(\bar{v}_j))$ でなければならない． （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，かつそのときのみ，$\bar{v}_i$ 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，$\pi(\bar{S}) = \{C_1, \cdots, C_m\}$，各 $C_i \subseteq \bar{V}$ とおくと，$C_1 \cup \cdots \cup C_m = \bar{V}$，$i \neq j$ ならば $C_i \cap C_j = \phi$ 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 $S \in R$ 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 $\tilde{S}$ 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つの手続きにわかれている。

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，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 x のサブエンドを求める。

ここでは，$\bar{S}$ の各節点 x に対し，x のサブエンド $E(x)$ およびレベル $\ell(x)$ 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 x に対してサブエンド $E(x)$ 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 i） 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 $[\bar{v}, 1]$，$[\bar{v}, 2]$，…，$[\bar{v}, l(\bar{v})]$ 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 $[\bar{v}, l(\bar{v})+1]$ を設け，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（$f(n)$ に比例する）の意味。

に対して，節点 $[v', k]$ が存在することは，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，$\ell(v) = \ell(v') + 1$ であることからあきらかである．

（ステップ ii）$[v, \ell(v)+1]$ 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，$[v_1, \ell(v_1)]$ と $[v_2, \ell(v_2)]$ が同じブロックに含まれているならば，v_1 と v_2 を同じブロックに入れて，S の節点を分割する（開始節点，終了節点は，それぞれ，そ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，$\pi(S)$ 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，$L(v, E(v)) = L([v, \ell(v)], [E(v), \ell(v)])$ となっている．また，ステップ ii の結果 $L([v_1, \ell(v_1)], [E(v_1), \ell(v_1)]) = L([v_2, \ell(v_2)], [E(v_2), \ell(v_2)])$ のときまたそのときにかぎって，$[v_1, \ell(v_1)]$ と $[v_2, \ell(v_2)]$ 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 $\pi(S)$ である．

ステップ i の手続きは $O(|V|^2)$ 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 $\tilde{S}$の定義

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

［手続き3-4］ $\pi(\bar{S})$の各ブロックに対応して，$\tilde{S}$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）．vが$\bar{S}$の演算節点のときには，$C(v)$を演算節点とし，vと同じ関数記号を付ける．vが$\bar{S}$の分岐節点のときには，$C(v)$を分岐節点とする（ラベルは付けない）．$C(v_I)$，$C(v_E)$を，それぞれ，$\tilde{S}$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vからv'へ枝があるとき，$C(v)$から$C(v')$への遷移を以下のように定める．

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

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

(iii) $\ell(v)>\ell(v')$のとき，節点$C(v)$からリターン節点に枝を結ぶ，vから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$\tilde{S}$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 $C(E^k(\tau'))$, …, $C(E(\tau'))$ がブロック $C(\tau)$ 内の他の節点に対してもそれぞれ同一になることが，$\pi(s)$ の定義と補題 3-3 から証明できる。(ii) 以外に関しては，$\pi(s)$ の定義からあきらか。（略証終）

§3.6　$\tilde{S}$の等価性とコスト最小性

$\bar{S}$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$\ell(\bar{v})$より小さくなく，$\bar{v}'$のレベル$\ell(\bar{v}')$は$\ell(\bar{v})$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$\ell(\bar{v})$とのレベル差がi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がなく，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i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］　$(\bar{v}, \bar{w}, \bar{v}') \in \bar{M}_i$であれば，$(C(\bar{v}), \bar{w}, C(\bar{v}')) \in \tilde{M}_i$がなりたち，また逆に，$(\tilde{v}, \tilde{w}, \tilde{v}') \in \tilde{M}_i$であれば，$\tilde{v} = C(\bar{v})$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$(\bar{v}, \tilde{w}, \bar{v}') \in \bar{M}_i$の関係がなりたつ．

この補題3-9はi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$\tilde{v}_I$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iに対して，$(\tilde{v}_I, \tilde{w}, \tilde{v}_E) \in \tilde{M}_i$

がなりたつ。$\tilde{v}_E = C(\bar{v}_E)$, $\tilde{v}_E = C(\bar{v}_E)$ 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 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。

S の各節点 v（スタック，リターン節点以外の）に対応する $\bar{S}$ の節点の集合を $\eta(v)$ 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］ S のスタック，リターン以外の節点 v に対して，$\eta(v) = \{\bar{v}_1, \bar{v}_2, \cdots, \bar{v}_n\}$ とすると，$\bar{v}_1, \bar{v}_2, \cdots, \bar{v}_n$ 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 S を考える。$\tilde{S}$ と $\bar{S}$ が等価であり，$\bar{S}$ が条件A1を満足することより，$\bar{S}$ の任意の節点 $\bar{v}$ に対して，$\bar{v} \in \eta(v)$ となるような S の節点 v が存在する。これと補題3－11より，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 v に対しての $\eta(v)$ を考えると，補題3－11，定理3－1より，$\eta(v)$ の全体は分割 $\pi(\bar{S})$ に一致する。

§3.7 結言

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）の多入口サブ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ができる手続きを示した．

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ブルーチンはサブルーチン化しなかったり，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ブ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，$\bar{v}_i \in \sigma(\hat{v})$ であることより，ある w_i に対して，$\hat{v}_I \xrightarrow{w_i} \hat{v}$，$\bar{v}_I \xrightarrow{w_i} \bar{v}_i$ がなりたつ（このことは以下で断わりなく使用する）

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

（i）　$\hat{S}$ は条件A2を満足しており“無駄”な節点を含んでいない．また，$\bar{S}$ と $\hat{S}$ は等価である．ゆえに，$\hat{S}$ における任意の $w \in T(\hat{v})$ に対して，$\bar{S}$ においては，節点 $\bar{v}_i \in \sigma(\hat{v})$ 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 $\bar{v}_i \in \sigma(\hat{v})$ から，系列 $w \in T(\hat{v})$ 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'，$w'' \in T(\hat{v})$ 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）．

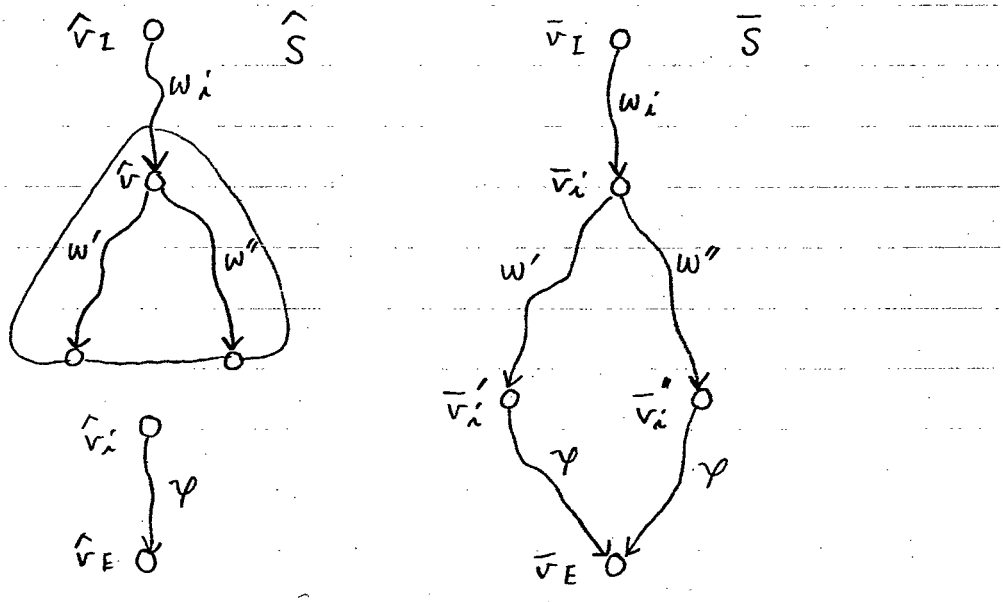


図 3－5　　補題 3－4－ii の証明の説明図

ゆえに，補題 3－2 より $\bar{v}_i'$，$\bar{v}_i''$ は同一節点でなければならない．

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

以下，$\bar{v}_i$ から $T(\hat{v})$ 内の任意の系列による遷移の行先の節点を $\bar{v}_i'$ と書く．

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

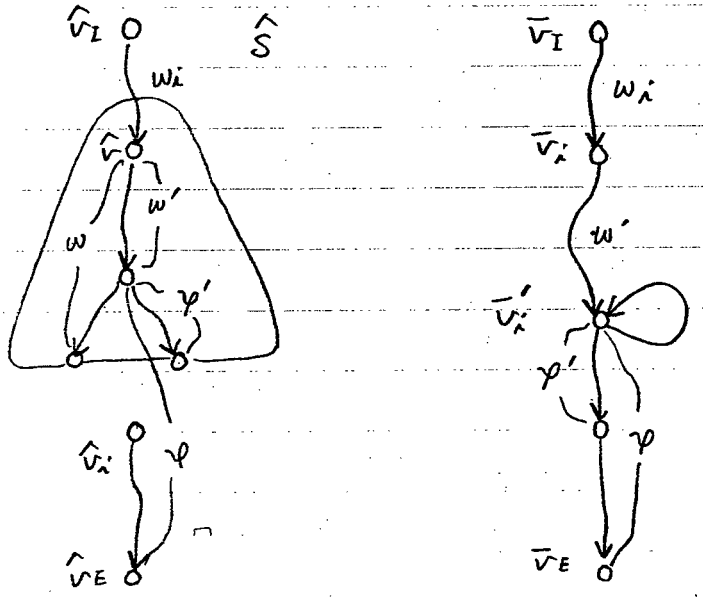


図 3－6　　補題 3－4－iii の証明の説明図

(iii の証明) $\bar{v}_i$ から $T(\hat{v})$ 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 A1 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 φ とすると，$w_i\ w'\varphi \in L(\bar{v}_I, \bar{v}_E) = T(\hat{v}_I)$ となる。$\hat{S}$ において，w' が $w \in T(\hat{v})$ の始系列であるため，$w'\varphi$ のある始系列 $w'\varphi'$（φ' は空系列でない）が $T(\hat{v})$ に含まれる。すると，(ii) より，$\bar{v}_i \in \sigma(\hat{v})$ から系列 $w'\varphi'$ 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 φ の始系列 φ' による遷移の行き先が $\bar{v}_i'$ となる。これは，φ 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) さもし，$\bar{v}_i$ 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。$w_i\ w \in L(\bar{v}_I, \bar{v}_E) = T(\hat{v}_I)$ より，$\hat{S}$ において，w のある始系列 w' が $T(\hat{v})$ 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，$T(\hat{v}) \subset L(\bar{v}_i', E^{k_i}(\bar{v}_i'))$ 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 A1 を満足しており，"無駄な" 節点を含んでなく，また，$\bar{S}$ と $\hat{S}$ 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，$\hat{S}$ において，遷移 $\hat{v}_I \xrightarrow{w_i} \hat{v}_i$（$\hat{v}_I \xrightarrow{w_i} \bar{v}_i$，$\bar{v}_i \in \sigma(\hat{v}_i)$）のあとに，$\hat{v}_i$ 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，$\hat{S}$ において，$w \in T(\hat{v})$ 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 $T(\hat{v})$ 内

に含まれるか，あるいは，wのある真の始系列w''が$T(\hat{v})$内に含まれるかのいずれかである。前者の場合，$T(\hat{v}) \subset L(\bar{v}_i, E^{k_i}(\bar{v}_i))$であることから，$ww' \in L(\bar{v}_i, E^{k_i}(\bar{v}_i))$がなりたつ。しかし，$w \in L(\bar{v}_i, E^{k_i}(\bar{v}_i))$でもあることから(1)と矛盾する。同様の方法で証明できる。

(証明終)

第4章　プログラムモジュール群の合併

§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[11]”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s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s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s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（つぎつぎ）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”におさまる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て済む），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”をそこまで許してもいわゆる“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，節点対(s, t)を“エフェクト枝”とも呼び，“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と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，もとの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 $v \to w$ と書く。グラフ G のニつの節点 s，t に対し，記法 $s \overset{*}{\to} t$ は，s から t へ枝の向きに沿ったパスが存在することを表わす（s，t が同一節点のときはつねに $s \overset{*}{\to} t$ が成り立つとする）。また，便宜上，$s \overset{*}{\to} t$ 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 $v_M \overset{*}{\to} v$ 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もつ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 $s \overset{*}{\to} t$ 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 $v \overset{*}{\to} v$ 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 $\pi = \{\beta_1, \beta_2, \cdots, \beta_l\}$ [+] 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 $v \to w$，$v \in \beta_i$，$w \in \beta_j$ なる節点 v，w が存在するとき，$\beta_i \Rightarrow \beta_j$ と書く．

この章ではつぎの問題を考察する．

［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．

〔非サイクリック条件〕 ブロック間の関係 $\Rightarrow$ によって閉路（$\beta_i \Rightarrow \cdots \Rightarrow \beta_i$）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”間のみに局限されるように，もとのモジュールを合併し，モジュール群を再構成する問題である．

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

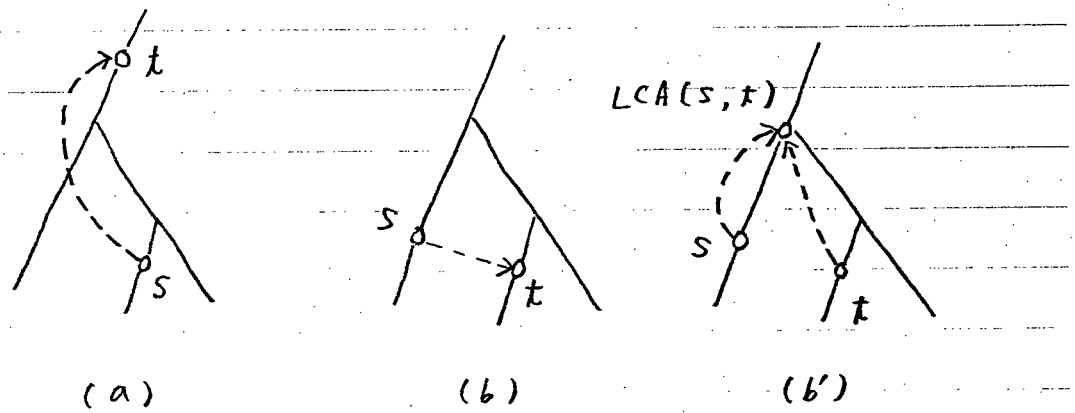
§4.3　Gが有向木である場合

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［連続条件を満足していれば，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，［連結条件］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）枝の集合を与えてもよい．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に属する節点対 (s, t) は，前述の仮定(ii)より，$t \to s$ 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)



(a)　　(b)　　(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し，(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，$v \to v_1$, $v \to v_2$であり，かつ，$v \to v'$, $v' \to v_1$, $v' \to v_2$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(いいかえれば，$t \to s$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$t \to s$が成り立つようなエフェクト関係Eのクラスを$\mathcal{E}$ backwardsと書く。もちろん，任意のEに対し，f(E)$\in \mathcal{E}$ 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$\in \mathcal{E}$ 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$\mathcal{E}$ 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($\in \mathcal{E}$ backwards)内の任意のエフェクト枝(s, t)に対し，G上のパス$t \to s$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できまる分割元は，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できまる分割を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$t \overset{*}{\Rightarrow} s$は$\Delta$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B_i，B_jとすると$B_i = B_j$または$B_j \Rightarrow B_i$であることを意味する。$(s, t) \in E$かつ$(s, LCA(s, t)) \in f(E)$，$(t, LCA(s, t)) \in f(E)$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，$LCA(s, t)$を含むπのブロックをそれぞれB_i，B_j，B_kとすると，B_iに関し$B_i = B_k$または$B_k \Rightarrow B_i$，B_jに関し$B_j = B_k$または$B_k \Rightarrow B_j$が成り立つ。上記のいずれの場合でも，B_iとB_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，パス $t \rightarrow s$ 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．$(s, t) \in E$ かつ $(s, LCA(s,t)) \in f(E)$, $(t, LCA(s,t)) \in f(E)$ であるようなEのエフェクト枝 (s, t) に対し，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，前述の［近接条件］中の(a)～(d)の一つがなりたつ。$LCA(s,t)$ 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 $LCA(s,t)$ 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 $LCA(s,t)$ 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 がなたして，$\beta'_h \Rightarrow \beta'_i$, $\beta'_h \Rightarrow \beta'_j$ の場合は，β'_h は $LCA(s,t)$ を含まねばならず $\beta'_k = \beta'_h$ でなければならない。以上のことより，パス $LCA(s,t) \rightarrow s$, $LCA(s,t) \rightarrow t$ 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対(s, t)に対し，文献(13)のアルゴリズムによりLCA(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4-1の手数は$|V|+|E|\times G(|V|)$のオーダである。

(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$B \in$ 匠 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，Δ(v)←φとする．（Δ(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の実パラメータを根v_Mにして実行する．Δ(v_M)が求まる．
3. Δ←Δ(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Δ(v)を求める手続き）

1. vが葉であれば，何もせずリターンする．
2. vが葉でないとき，

   v→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，

   Δ(v)←Δ(v)∪Δ(w)∪{(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）

   Δ(v)←Δ(v)∪Δ(w)

3. v→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, CONTAIN(w, Δ')を, もし t →* w, w →* s かつ w →* 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, UNCONTAIN(w, Δ')を, もし t →* w, w →* s かつ w →* 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 → wとし, 今, Δvを[カット条件]を満たしているGv内の枝の部分集合とするとき, Δvのうち Gwに属する枝のみでできる集合をΔw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)

$$
\begin{aligned}
&CONTAIN(v, \Delta v)\\
&= MIN\,[\{CONTAIN(w, \Delta w) \mid v \to w,\ (v, w) \notin \Delta v\}\\
&\qquad \cup \{UNCONTAIN(w', \Delta w') \mid v \to w',\ (v, w') \in \Delta v\}\\
&\qquad \cup \{DEPTH(v)\}] \qquad (1)\\
&UNCONTAIN(v, \Delta v)\\
&= MIN\,[\{UNCONTAIN(w, \Delta w) \mid v \to w,\ (v, w) \notin \Delta v\}\\
&\qquad \cup \{DEPTH(t) \mid (v, t) \in B\}\\
&\qquad \cup \{DEPTH(v)\} \qquad (2)
\end{aligned}
$$

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うつる. 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

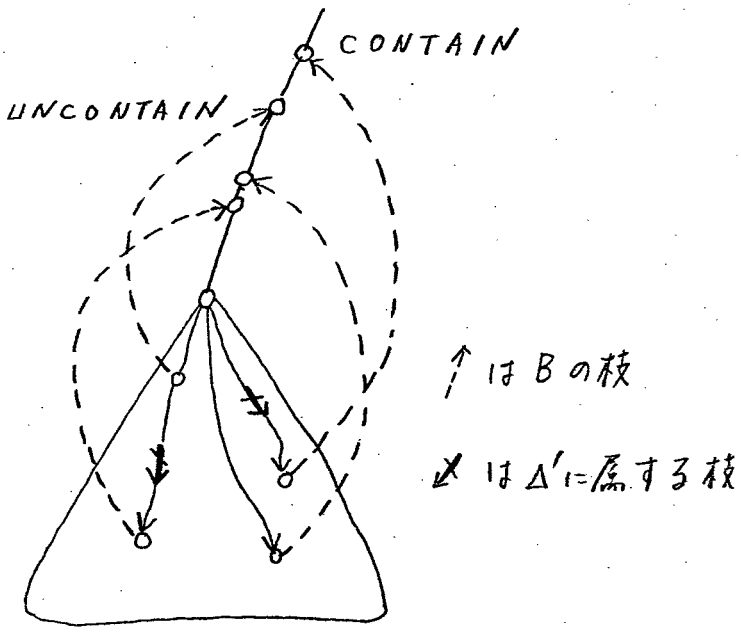


図4-2　　CONTAIN, UNCONTAIN の定義

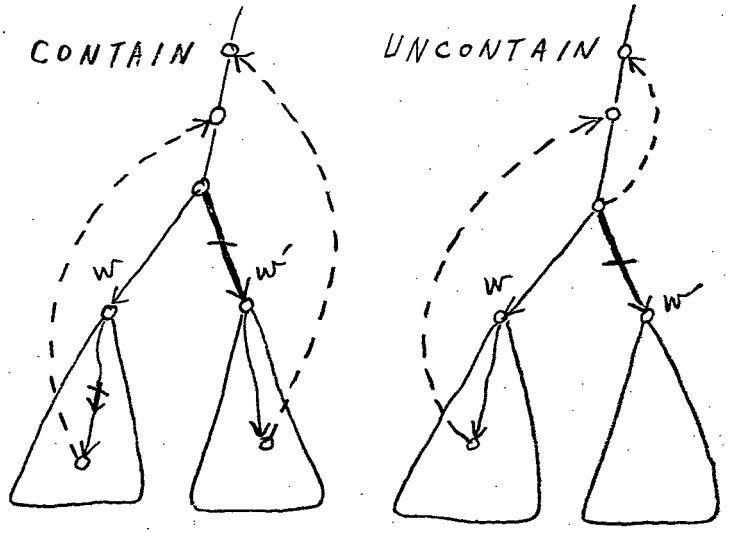


図4-3　　CONTAIN, UNCONTAIN の性質

［補題 4-3］ 前述のΔを求める手順における手続き CUTEDGE(v)によって求められたGv内の枝の部分集合Δ(v)は，Gv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，そのうちでも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，Gvに属する枝はないので，Δ(v)=φ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→wである各節点wに対しΔ(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．

Δ(v)が［カット条件］を満たすことは，手続きCUTEDGE(v)より，あきらかである．

Δ(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，Gv内の枝のみからなるカット枝の集合Δvで，(イ)［カット条件］を満足し，かつ|Δv|>|Δ(v)|なるもの，または，(ロ)［カット条件］を満足し，かつ|Δv|=|Δ(v)|であるが，CONTAIN(v, Δv)>CONTAIN(v, Δ(v))なるものが存在すると仮定する．

v→wなるwに対し，Δvのうち Gw内に属するカット枝集合Δwで表わす．Δvが［カット条件］を満たしていれば，Δw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→wなる各wに対し，|Δw|≦|Δ(w)| ゆえ，|Δv|>|Δ(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：$v \to w$ なる w のなかに，$|\Delta w'| = |\Delta(w')|$ で，かつ $(v, w') \in \Delta v$，$(v, w') \notin \Delta(v)$ であるような w' が在在する。]

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，$\Delta w'$ が[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，$CONTAIN(w', \Delta w') = DEPTH(w')$。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，$CONTAIN(w', \Delta(w')) < DEPTH(w')$ であったことを意味する。ゆえに，$CONTAIN(w', \Delta(w')) < CONTAIN(w', \Delta w')$ 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，$CONTAIN(v, \Delta(v))$ の値は，(a) $(v, w) \notin \Delta(v)$ であるようなある w に対する $CONTAIN(w, \Delta(w))$ か，(b) $(v, w) \in \Delta(v)$ であるようなある w に対する $UNCONTAIN(w, \Delta(w))$ か，(c) $DEPTH(v)$ かのいずれかと等しくなる。

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

この二つの式は矛盾するので，$(v, w) \notin \Delta v$ 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わけて考える。

(a, 1) $|\Delta w| < |\Delta(w)|$ の場合，(a) の場合の条件と上記の結果より $(v, w) \notin \Delta(v)$，$(v, w) \notin \Delta v$ 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，Δw 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 $CONTAIN(v, \Delta v) > CONTAIN(v, \Delta(v)) = CONTAIN(w, \Delta(w))$ であるが，一般に CONTAIN の定義より $CONTAIN(w, \Delta w) \geq CONTAIN(v, \Delta v)$ ゆえ，$CONTAIN(w, \Delta w) > CONTAIN(w, \Delta(w))$ 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，もし $(v, w) \in \Delta v$ と仮定すると，$CONTAIN(v, \Delta v) \leq MIN\{DEPTH(t) \mid (s, t) \in B, t \overset{*}{\rightarrow} v, w \overset{*}{\rightarrow} s\} \leq UNCONTAIN(w, \Delta(w))$ となり，一方，(ロ) および (b) の場合の条件より $CONTAIN(v, \Delta v) > CONTAIN(v, \Delta(v)) = UNCONTAIN(w, \Delta(w))$ であるが，この二つの式は矛盾するので，$(v, w) \notin \Delta v$ 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，$(v, w) \in \Delta(v)$，$(v, w) \notin \Delta v$ 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, Δ(v)) = DEPTH(v)は，CONTAIN の定義および（ロ）の場合の条件より得られる式 DEPTH(v) ≧ CONTAIN(v, Δv) > CONTAIN(v, Δ(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 Δ(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, Δ(w)) を用いて，

CONTAIN(w, Δ(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，Δ(w) は［カット条件］を満たしているので，CONTAIN(v, Δ(v)) の値は，Δ(v) を求めるとき，前述の式(1)，(2)によってつぎつぎ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 Δ(v) を求める手続き CUTEDGE(v) を CONTAIN(w, Δ(w)) および UNCONTAIN(w, Δ(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, Δ(v)), UNCONTAIN(v, Δ(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，Δ(v), CONTAIN(v, Δ(v)), UNCONTAIN(v, Δ(v)) の初期値として，それぞれ，φ，DEPTH(v)，MIN［{DEPTH(x) | (v, x) ∈ B} ∪

{DEPTH(v)}]をセットしておき，再帰的手続き CUTEDGE(v) 内において，Δ(v)，CONTAIN(v, Δ(v))，UNCONTAIN(v, Δ(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 Δ(v) は枝の集合を表わしているが，ポインタでつなぐようにすれば Δ(v) ← Δ(v) ∪ Δ(w) ∪ {(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| のオーダの手数を要し，また，各節点 v に対し，CUTEDGE(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手数で済む．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| のオーダの手数で求まる．

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) の枝数はもとの 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| × G(|V|) のオーダの手数で求められる．

```
input : 有向木G = (V,A), エフェクト枝の集合 B ∈ 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 ∈ V
     begin
       Δ(v) ← φ ;
       CONTAIN(v,Δ(v)) ← DEPTH(v) ;
       UNCONTAIN(v,Δ(v)) ← MIN[{DEPTH(t)|(v,t) ∈ B}
                                ∪{DEPTH(v)}] ;
     end
   call CUTEDGE(v_M) ;
   Δ ← Δ(v_M) ;
 end

procedure CUTEDGE(v)
 begin
   if v is a leaf then
     return ;
   else
     begin
       for each w such that (v,w) ∈ A
         begin
           call CUTEDGE(w) ;
           if CONTAIN(w,Δ(w)) = DEPTH(w) then
             begin
               Δ(v) ← Δ(v) ∪ Δ(w) ∪ {(v,w)} ;
               if UNCONTAIN(w,Δ(w)) < CONTAIN(v,Δ(v)) then
                  CONTAIN(v,Δ(v)) ← UNCONTAIN(w,Δ(w)) ;
             end
           else
             begin
               Δ(v) ← Δ(v) ∪ Δ(w) ;
               if CONTAIN(w,Δ(w)) < CONTAIN(v,Δ(v)) then
                  CONTAIN(v,Δ(v)) ← CONTAIN(w,Δ(w)) ;
               if UNCONTAIN(w,Δ(w)) < UNCONTAIN(v,Δ(v)) then
                  UNCONTAIN(v,Δ(v)) ← UNCONTAIN(w,Δ(w)) ;
             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]

§4.4 結言

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 $|V|+|E|\times G(|V|)$ のオーダの手数で求められることを示した。

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，高速のインタフェイ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，多入口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，$|V|^2 \log |V|$ 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，多入口サブルーチ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，できるだけもと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，合併すべき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中にどのように記述するかが問題となるが，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章で述べたデバイスシェアリングシステムは既なのDOSを改造することによって作製されたが，内部処

理を知る方法はリスティングを解読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読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る。この記述様式に関してはいくつかの提案があるが[14]、これらの記述様式に対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，前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斉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 programer'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).