解法を表すプログラム図法の実現性分析†

佐藤 匡正††

プログラム図法はプログラム生産性改善には重要な役割を果たす．プログラムを分かりやすく表現できれば開発チーム内の意思伝達がよくなり誤りの防止と早期発見に役立つ．ここでは，解法とプログラムを一体化するための図法の考え方とその実現性について論ずる．従来から使われているフローチャート図法などの手順式記法による図面ではプログラムか解法かの一方しか表せない．これでは，分かりにくい．解法を中心とすると，機能中心となるのでプログラム細部の規定が不明確になる．プログラムを中心とすると，解法が見えなくなり分かりにくい．解法とこれを実現しているプログラムとの対応を明快にすれば分かりやすくなる．ここでは，解法をプログラム構造に整合させプログラムと一体化するという考えに立つ．解法をプログラムの構成原理である手順に合わせプログラムに取り込む．この操作によって得られた解法は木構造型の階層構造となる．この構造では，プログラムは層を単位としてその流れが層内に閉じるように区切られている．本論文では，上の解法と手順を一体化するという着想の実現性を示すとともに，この実現のための基本的な方策—手順の抽象化とデータ記述—を論じ，技法として具体化した場合に予想される得失を考察し，この着想が有効であると結論づける．

1. はじめに

プログラムの開発を支援する重要な技法のひとつにプログラム文書化技法がある．この技法はプログラムを表すためのもので，プログラムの記述（コーディング）を始めとして，作業進捗管理や保守，教育などに幅広く使用されている．この代表的な技法としてフローチャート図法（以下，FC 図法）がある．しかし，これは有効とはされていない．もともと，図がプログラム言語より優位なのは記述水準が高い点にある．ところが，構造化プログラミング技法の概念とともに普及した高水準言語には構造化するための機能—データ構造や整理された制御構造—が備わっており，記述水準が向上している．この向上分だけ，FC 図法での記述水準が相対的に低下している．

一方，プログラミング以外の分野では回路図や建築の図面など，設計図という言葉が日常化しているほど設計においては図が重用されている．これは設計の内容が図できちんと書け，分かりやすさが他の記法より優れているためと考えられる．プログラミングの分野において，この図のもつ特徴をうまく活用できるような書き方が編み出せれば，図面がプログラムの設計書として再認識される．分かりやすい図が書ければ，読者は自分だけに閉じず他人にも広がる．これによって，作成チーム内に意思が伝わりやすくなり誤りの防止や早期発見などに役立ち，プログラムの作成過程が円滑に進められる．

最近では，FC 図法の低下した記述水準を上げるべく構造化の概念を導入した木構造チャートとか構造化チャートとか呼ばれる改善提案がなされている[5),6)]．これらの提案は基本的には記法の改善である．プログラムをどのように書くかを改善するための提案が中心であり，プログラムとして何を書くべきかの議論は十分とはいえない．分かりやすい図を書くためには，どのように書くかの前に，何を書くべきかを論ずる必要がある．何を書くべきかがはっきりした後に書き方を決めるのが順序である．

ここでは，手順を主体とするプログラム図法を分かりやすくするための着想として，これまでは暗に規定されていた解法を，明に書くことを提案する．これを図法として具体化するには，この解法とこれまで明に規定されていた手順を一体化した新記法を実現しなくてはならない．本論文では，この図法の実現性および，得失について議論する．なお，本論文においては，記法による偏りを避けるために最も基本的な FC 図法を用いる．

2. 着想—'what と how'

プログラムを分かりやすく表すためには図面に何を書いたらよいか．FC 図には，明にプログラム動作が，暗にその問題が規定されている．この「暗に規定されている問題」がきちんと書かれれば分かりやすくなる．

FC 図の「暗の規定」を例示する．このために，FC 図のプログラムを理解するまでの過程を振り返ってみ

† A Feasibility Study on an Algorithm-Program Documation Technique by TADAMASA SATOH (Electrical Communication Laboratories, Nippon Telegraph and Telephone Corporation).

†† NTT 情報通信処理研究所

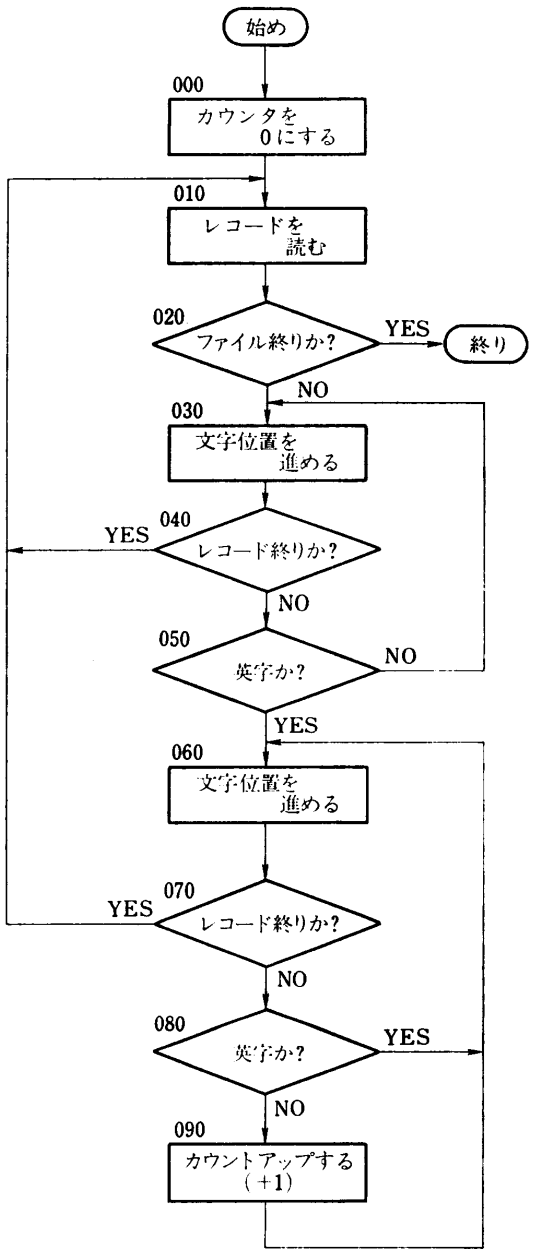


図 1 フローチャート図の例題
Fig. 1 An example of conventional flow charting.

る．**図 1** に簡単な例題を与える．例題の問題記述は今は与えないでおく．このプログラムを理解するには，図の処理ボックスを，実行順序を表す線のつながりに従って読み進んで行く．ボックス間の関係に注意しながら流れに区切りをいれて，その「まとまり」全体で何をしようとしているかを推理する．例えば，「文字位置を進める」→「英字か？ 英字でなければ繰り返す．英字なら繰り返しを止める」という連なりを追い，文字列で英字が現れるまで文字を読み飛ばしているのであろうと見当をつける．これから「単語の始まりをみつける」を目的とする処理であると理解する．つまり，FC 図法を理解する過程は，どのように処理を行うかを追跡してから，何がやられるかを推理し理解する．この「何」が暗の規定部分であり，「どのように」が明の規定部分である．この「何」を 'what'，「どのように」を 'how' ということにする．what は細分化された一つの問題と捉えることができる．what が明記されていれば問題が構造体として規定されるので分かりやすくなる．

そこで，試しに what を書き入れた FC 図を作ってみる．what は細分化された一つの問題であり，いくつかボックスの処理目的を指すので，これを「(A)を(B)する」という文で表す．図式ではこの what を仮に，平面で表してみる．FC 図中の関係のある処理ボックスをまとめて一つの平面で区切り，これに what を与える．このようにして得られた平面を単位としてさらに関係をもつ平面をまとめて上位の平面を作る．この措置を図のすべてについて行う．この平面の中に書かれている手順は，what に対する実現手段，つまり，how を表す．こうすると，この平面を単位とした木構造型の階層をなす図が得られる（**図 2**）．

図の最上位の階層（P面）では，問題を what で大まかに表している．つまり，「ファイル内の単語数を数える」である．次に続く階層（I面）ではこの what を次位の what と how（手順）で実現している．I面での次位の what は「レコードを読む」，「レコード内の単語数を数える」であり，how は「ファイルが終わるまで繰り返す」である．この平面は階層が深まるに従い段々と詳細になってゆく．最終的にはコードレベルに到達する．こうして得られた図 2 は (what, how) の組を単位とした階層構造を成している．これは，トップダウンに読むことができるので図 1 よりは分かりやすい．図 1，2 の比較から，FC 図がプログラムの明の規定しか表していない様子が分かる．図 2 の立体的な構造が図 1 では無構造な一つの平面に縮退している．図 1 は，コーディングとして実体をもたない暗の規定（図 2 では点線の箱で示してある what の部分）が欠けている．

さて，文献 1) では解法を，順序，選択などの機能関係で構成した機能の構造体として捉えた．ここで，機能は（出力，操作，入力）の組の文で表される．what を機能に相当するといえれば，上で述べた「暗の規

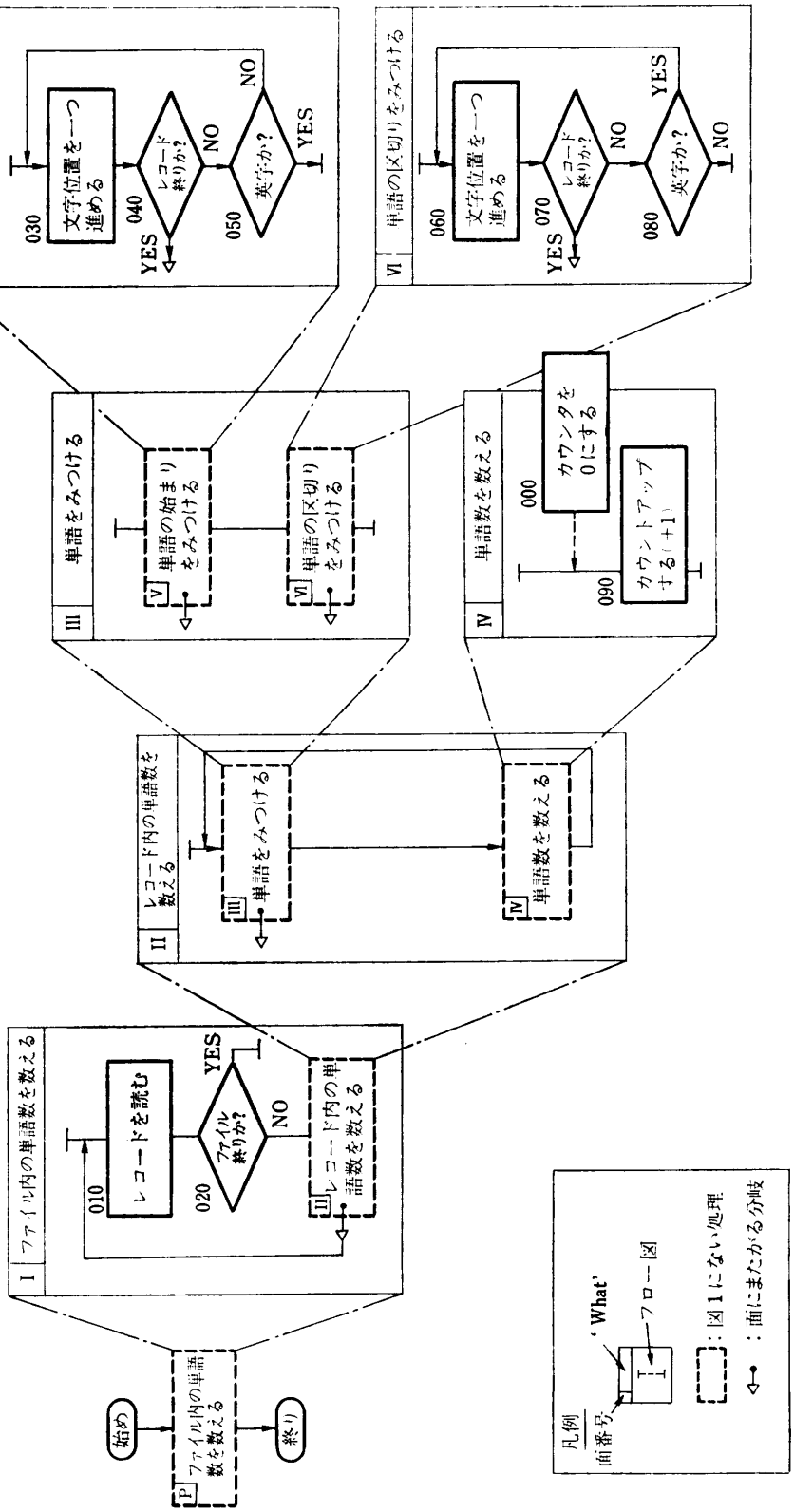


図 2 図1の 'what' 階層図

Fig. 2 A 'what' hierarchical chart (from Fig. 1).

定」は解法に相当する．what を表す文において，(A)には変数を，(B)には操作をあてはめれば，出力，入力は空も含まれるので what は機能に比定できる．こうすると，図2の what 平面同士は基本的には機能を手順で構成したものであるので「暗の規定」は解法を表している．この手順は繰り返しなどは無く連接だけである．これは解法における機能関係を順序に限定したものに相当する．これから図2での改善理由が分かる．what を書くことによって，通常の FC 図法での手順処理に解法が取り込まれたためである．

3. 解法とプログラムの合体

解法は機能関係とデータによって機能が結ばれたものである．ここでの解法は機能分析図で表されているものと捉える．プログラムはデータ操作の手順である．データ操作は機能に含まれ，機能関係は抽象化した手順に含まれる．よって，解法とプログラムは抽象化した手順とデータを規定することによって合体できる．ここでは，合体するための，手順の抽象化の方法およびデータの表現方法について述べる．

3.1 手順の抽象化

(1) 定形手順と非定形手順

プログラムを一つの平面で区切るには手順を一入口，一出口に限定すればよい．このためには，プログラムの手順をよく知られている三つの制御構造，連接，繰り返し，選択に整理すればよい．この手順を定形手順と名付ける．基本的には，定形手順は解法の機能関係と対応がとれている．一方，エラーのあと処理などの手順は定形化し難い．フラグなどを用いて基本制御構造に変形することも不可能ではないが，こうすると後述（(3)項）するように解法のもつ機能関係を壊すことがある．ここでは，定形化の変形をしないで扱う．この手順を非定形手順と名付ける．通常は，非定形手順は解法として明記されない．

(2) 制御と仕掛け

ある面内の手順には，機能関係に対応している部分機能の結び付きを制御するものとこの制御の仕掛けとが混在している．ここで，部分機能の「部分」という用語は階層の「下位」を表す接頭語である．制御は連接，繰り返し，および選択であり，解法と対応している．仕掛けは繰り返しや選択を実現している．解法とは間接的にしか対応していないプログラム上の規定である．そこで，機能平面においては，解法と対応している部分機能および制御を主とし，対応していない仕掛けを従と区別すると整理される．こうするためには，仕掛けを抽象化しかつ局所化して制御との対応を明快にすることが大切である．

図3(A)の FC 図では，文字列中の数字を数えるプログラムの一部を取り出したものである．この FC 図を図2のように平面で区切る(図(B))．このII面「二個以上なら警告する」における部分機能は「警告する」，または「警告しない」であり，解法と直接的に対応している．この部分機能の制御は選択であり，条件付は「…一個か」「…二個以上か」である．解法にはこの選択をどうやって実現するか，つまり仕掛けは規定されていない．プログラムとしては，図3(A)の点線で囲んだフラグのオン／オフで構成されている．この仕掛けの抽象化，局所化した表現例を図3(B)に示す．この例では，同図(A)の点線で囲んだ選択の仕掛けを二値判断記号◇の中に閉じ込めている．この結果，解法の条件付と機構の二値判断との対応がはっきりする．

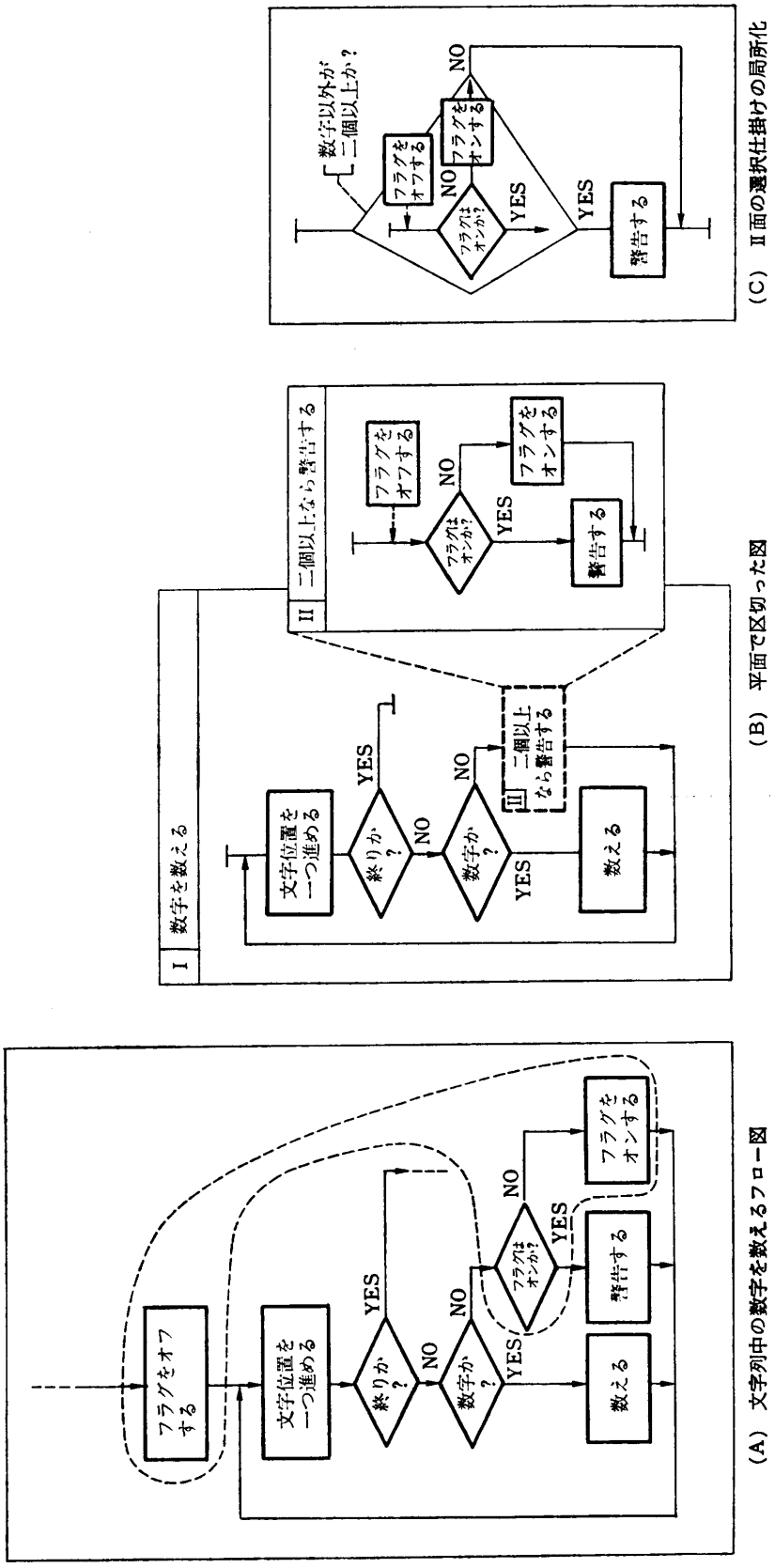
繰り返しの仕掛けは，回次（一回分の繰り返し）を進めることと終了条件を与えることから成る．このためのデータとしてカウンタやポインタが使われる．制御には，繰り返しの対象と終了条件が与えられれば十分である．仕掛けは選択の場合と同様に抽象化，局所化によって制御と対応づける．

(3) 非定形手順の仕掛け

非定形手順の影響によって定形手順が壊れることがある．解法と対応のとれている定形手順の形が壊れないように整理する．定形手順を主，非定形手順を従として規定する．

一般に，エラーの後処理などの非定形な手順は解法としては規定されていない．定形手順の部分機能が解法の本質である[2]．この定形手順に非定形手順を書き加えることにより本来の定形手順が壊れることがある．本質が変わってしまっては都合が悪い．非定形手順を従として区別するためには，部分機能とは別な記法，例えば文章で書く．

図4の FC 図は入力として得た文字列がファイル名として妥当かどうかをチェックしている．ファイル名の文字長が8より大きいとエラーである．エラーを見つけ，エラーメッセージを示した後に上方向に流れ線が延び再試行のための繰り返しを形作っている．本来の手順は繰り返しではない．本来の手順はエラー後の再試行のための繰り返しを除いた連接である．これは解法と対応している．促進メッセージを出力し，文



(A) 文字列中の数字を数えるフロー図

(B) 平面で区切った図

(C) II面の選択仕掛けの局所化

図 3 選択型定形手順の仕掛け

Fig. 3 A mechanism for the selection fixed type procedure.

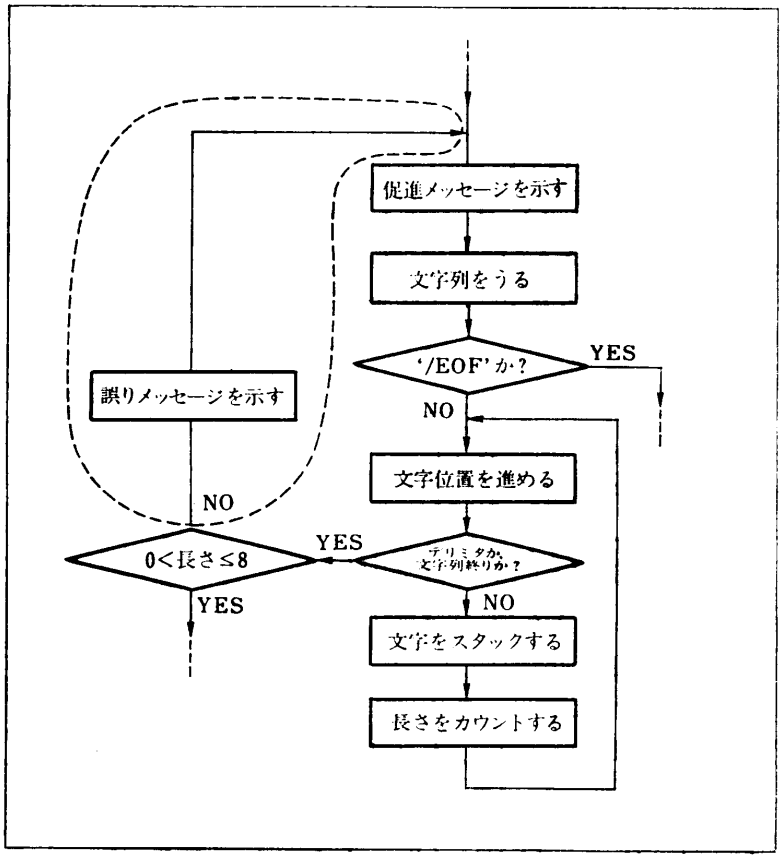


図 4 非定形手順の隠ぺい
Fig. 4 Hidden unfixed type procedure.

字列を得て，ファイル名をチェックする．記述する場合には，本質であるこの定形手順を主に，非定形手順であるエラーの後処理用繰り返しの流れは従となるようにして区別する．例えば，〔長さが0または9以上〕の条件では，エラーを意味する記号を設け，後処理の記述は，「再試行する」と表示するにとどめる．

（4） 階層の飛び越し

平面の飛び越しを面に閉じさせるように，飛び越す階層数と平面における飛び越し先を一意に規定する．図2では，V，VI面で飛び越しがある．[レコード終わり]で条件づけられた流れはIII面を飛び越し，I面の部分機能「レコードを読む」に続く．面の結びと結ばれる部分機能を規定する．

3.2 データ表現

機能の要素であるデータはよく知られているように構造をもつ．この構造は，プログラムの階層，連接，繰り返し，選択と対応する[4]．したがって図面中にデータを書くことによってプログラム構造の見通しが利き，抽象化を高められる．データのこの特徴を生かすために次の働きをもたせる．

（1） データ構造の抽象化

構造を抽象化する．例えば，ファイルのレコード，配列，ブロック列は抽象化して同一の繰り返しデータ構造とする．

（2） 機能間の結び付き

抽象化されたデータ操作（参照／設定）を介した機能間の結び付きによってデータ流を表す．さらに，初期値設定のような簡単な操作はプログラムの方ではなく，データの方に記述することによって，初期値設定操作の離散化の問題[3]は対処できる．

（3） 用途の区別

プログラム内での機能の結び付きか，外部との結び付きかを区別する．作業用と連絡（インタフェース）用かの種別を設ける．

4. 考　　察

従来の手順による記述に加えて解法として手順の抽象化とデータを盛り込んだ図法の主な狙いはチーム内の意志伝達の円滑化である．ここでは，これ以外に期待される効果を推定し，併せて問題点を議論する．

4.1 期待される効果

（1） 他の図法との組み合わせ

状態遷移図(STD図)との対応が明快である．STD図のアークは平面と対応している．図2の問題を例として対応を示す．この問題では入力例として，レコード列とレコード内文字列の二つを含むので二つのSTD図ができる．図2の面との対応を示すために，STD図を枠で囲み，面番号（I～VI）を付ける．図5に示す．図ではI面でレコード列のSTD図，II～VI面で文字列のSTD図を示す．文字列のSTD図では，V，VI面の二つの基本STD図がII，III面でのアークによって連結され，機能を表している．III面ではV→VIの連結で単語をみつけることをII面では，V←VIでIII面の反復，つまりレコード内の単語をみつけることを，表している．このことは3章で提案した階層化により，STD図だけでなく他のより

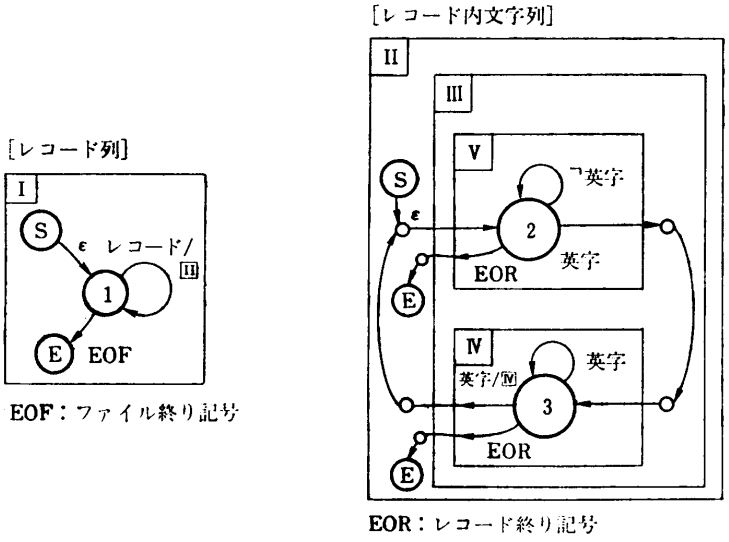


図 5 図2とその状態遷移図
Fig. 5 Figure 2 and its state transition diagram.

抽象化の高い技法と円滑に組み合わせられ，仕様からコードまでの対応が明快に示せることを意味している．

(2) 抽 象 化

① 標 準 化

データや機能の抽象化の書き方を活用して，プログラミング言語の制御の抽象化や，データの抽象化の概念[7]に応じた標準化が図れる．例えば，ファイルのレコード列，配列，ポインタでつながるブロック列をアクセスする．これらの列を抽象化し「繰り返し」データ構造と捉えると操作も抽象化できる．レコードを読む，添字に1加える，ポインタを進めるなどの多様な操作表現を，「要素を得る」のように三者に共通とした抽象化が可能となる．この抽象化によって三者は同一の規定で済む．

② 機能の型の抽象化

定形手順を持つ機能は型によらず機能としての扱いを同一にできる．例えば，図6では，「文字列の長さを得る」という機能が選択型(A)，繰り返し型(B)の機能で示されている．部分機能の手順で示されている解法は A,B で異なるが機能は同一である．

(3) 不要な詳細の隠ぺい

非定形手順によって自明である手順の仕掛けが隠ぺいできる．また，本質についても，階層平面の最低水準を決めることによって開発チームのスキルに応じた詳細化の水準が設定できる．これにより，本質が浮き彫りになる，記述量を減らせる，修正の局所化が図れるなどの効果が生ずる．

4.2 議 論

短所と考えられる事項を列挙し，その対策を論ずる．

[1] 記述量が増える．ゴチャゴチャして分かりにくくなる．

解法やデータをこれまでの記述に加えるために一枚の図面に書く量は増えるが，従来は別の形式で書かれていたものを取り込む．したがって，重複した部分が無くなるので全体の記述量は減る．また，不要な詳細の隠ぺいによる記述量の減少も見込まれる．

[2] ぎっしり詰め込んだ図になる．修正，変更が

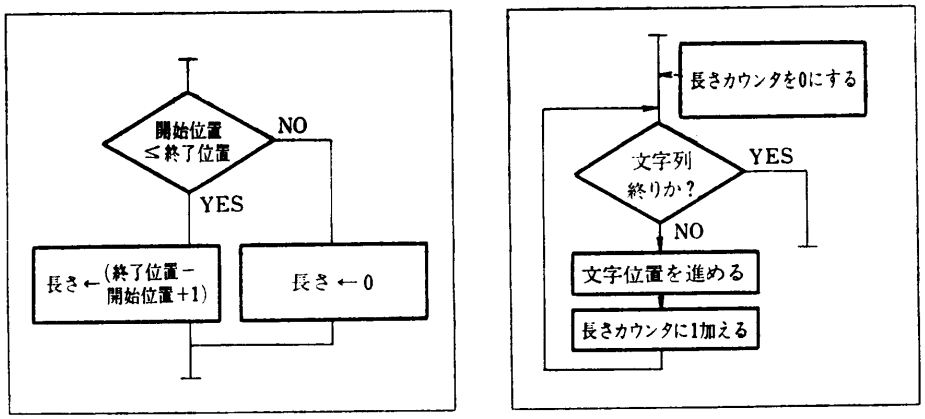


図 6 機能の型の抽象化
Fig. 6 Function-type abstraction.

大変である.

抽象化による記述水準の向上分だけ，① コードの変更には強い，② 変更箇所がひとつの層または，配下の層だけに限定される，という性質がある．根本的な対策は計算機による補助である.

［3］ 書き方が難しくなる.

ここでは具体的な書き方には触れていないが，書き方によっては慣れるのに時間がかかることになる．単純な原則の，従来と親和した記号を考案し，導入方法を工夫することによって円滑化できる．例えば，下書きとして，よく慣れている流れから書いてゆき，これに解法を上乗せし，整理する．最後にデータを加えて清書する，というのも一法である.

［4］ 書く人によって異なる図にならないか.

一般には，抽象化により図はむしろ標準化の方に向かう．違いの生ずる原因として，詳細化の水準や解法の違いがある．詳細化水準はある範囲で自由に設定できる．この水準の差異は見かけの違いである．図の違いは人や図法によるというよりもむしろ，解法によると考えられる．解法が異なれば図は当然異なる．解法の標準化はプログラムの書き方とは別の問題として取り組む必要がある.

5. おわりに

プログラムを分かりやすく書くと開発チーム内の意思伝達がよくなり開発が円滑に進む．分かりやすく書くには，これまで暗に規定されている解法と明に規定されているプログラムを対応づけるのが効果的である．この考えの実現性を考察した.

この考えによる図法では，抽象化による不要な詳細が省略できるので記述量を減らすことができるが，修正や導入が難しいという欠点がある．しかし，修正については計算機の支援によって解決でき，導入については従来技法と親和させた方法の工夫によって円滑化できる．以上により，これまでは暗に規定されていた解法を，明に規定されているプログラムに対応づけて表せる図法について実現性の見通しが得られた.

参考文献

1) 花田收悦，佐藤匡正，松本匡通，長野宏宣：コンパクトチャートを用いたプログラム設計法，情報処理学会論文誌，Vol. 22, No. 1, pp. 44-50 (1981).
2) 佐藤匡正：本質処理を出発点とするプログラム設計方法，情報処理学会「プログラム設計技法の実用化と発展」シンポジウム講演集，pp. 59-68 (1984. 4).
3) 佐藤匡正，浅見秀雄：フローチャート階層的表現のための一提案，第 20 回情報処理学会全国大会論文集，pp. 285-286 (1979).
4) 佐藤匡正，浅見秀雄：処理の論理構造と階層化チャート技法 HCP，第 23 回情報処理学会全国大会論文集，pp. 407-408 (1981).
5) 佐藤匡正：プログラミング用ドキュメンテーション，情報処理，Vol. 22, No. 5, pp. 383-389 (1981).
6) 青山義彦：ソフトウェア開発に用いられる図形表現法，情報処理，Vol. 25, No. 5, pp. 451-461 (1984).
7) Liskov, B. et al.: Abstraction Mechanisms in CLU, *CACM*, Vol. 20, No. 8, pp. 564-576 (1977).

(昭和 61 年 4 月 7 日受付)
(昭和 61 年 8 月 27 日採録)

佐藤 匡正（正会員）

昭和 42 年横浜国立大学工学部電気工学科卒業．同年日本電信電話公社に入社．電気通信研究所，データ通信本部などにおいて銀行業務処理システム，言語処理プログラムの開発に従事．現在，日本電信電話（株）から日本情報通信(株)に出向．プログラム設計方法とプログラム構造との関係に興味をもっている．情報処理学会論文賞受賞（昭和 56 年度）．電子通信学会会員.