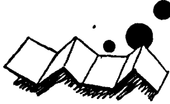
解説

プログラミング方法論（ジャクソン法）†



峰尾欽二††

1. はじめに

M. A. Jackson の提唱するジャクソン法（Jackson Structured Programming, 通称 JSP）は，主として事務計算分野で採用されているプログラミング方法論のひとつである．実際欧州の事務計算プログラミングの現場で広く活用されている．

この解説では，まず簡単な例題によって JSP の着想をしめし，ついで JSP の基本部分を説明する．この基本ジャクソン法をそのまま適用することが困難な問題にたいする一二の工夫をつぎに紹介する．これは後戻りと擬似コルーチン化である．また，ジャクソンはその方法を支援する Cobol の前処理系 JSP-COBOL を用意しているので，これを用いたコード化例をつけて，あわせてこの JSP-COBOL を紹介する．結びとして JSP の評価を与える．

2. ジャクソン法とは

JSP は，プログラムへの入出力データに注目し，データの構造を考察して，そのデータの構造からプログラムの構造を導く，という着想にもとづいている．これは，他の多くのプログラムの構造化設計法が行っているように，プログラムに要求される機能からその実現方法を直接考案する方法とは異なっている[1]．

まず簡単な例題を考えよう．

「顧客ごとに整列された売上データを入力し，顧客ごとの売上合計を印書する．なお，売上集計表の最初に表の名前を，最後に全顧客の売上合計を印書する．」

この課題をつぎのように考える．扱うデータは，入力，出力ともに順編成ファイルである．入力ファイルは顧客ごとのデータの繰返しであり，顧客ごとのデータは売上データである売上レコードの繰返しである．これを句構造文法にならって書くと，

<sales file>→<sales data> eof

<sales data>→<customer data>*

<customer data>→sales record*

が得られる（ここで a* は a の 0 回以上の繰返しを意味する．また <b> は b が非終端記号であることを，< > がないものは終端記号であることを示す）．ここで，入力ファイルは，売上レコード sales record と eof (end of file) を終端記号とする句構造言語の文であると考えている．出力の売上集計表は，同じように考えて，

<sales report>→title <report body> total

<report body>→line*

と書くことができる．ここで問題のプログラムを句構造言語 <sales file> から同様の <sales report> への翻訳プログラムであるとみなすことにすると，つぎのような翻訳文法を考えることができる．

《make report》→[title]《make body》eof[total]

《make body》→(《make line》)*

《make line》→sales record* [line]

（[ ] は出力側の終端記号を示す．）これをつぎのように解釈する．翻訳は《make report》の実行で始まり，まず title をつくり，つぎに《make body》を行い，ついで入力 eof に対して total をつくることである．《make body》は《make line》の繰返しである．《make line》は入力 <customer data> である sales record* に対して line をつくる．

この翻訳文法をわれわれが求めようとするプログラムの構造であると考える．ここで入力や出力のデータを正規言語とみなすことができる場合に対して，以上のような着想をジャクソンは以下に述べるような方法として手順化した．これが基本 JSP である[2)~4)]．

3. 基本ジャクソン法

基本 JSP はつぎの 4 段階の手順から構成されている．

1° データ構造の定義（データ・ステップ）

† Programming Method (Jackson Structrued Programming) by Kinji MINEO (NIPPON UNIVAC KAISHA, LTD.).

†† 日本ユニバック(株)教育部

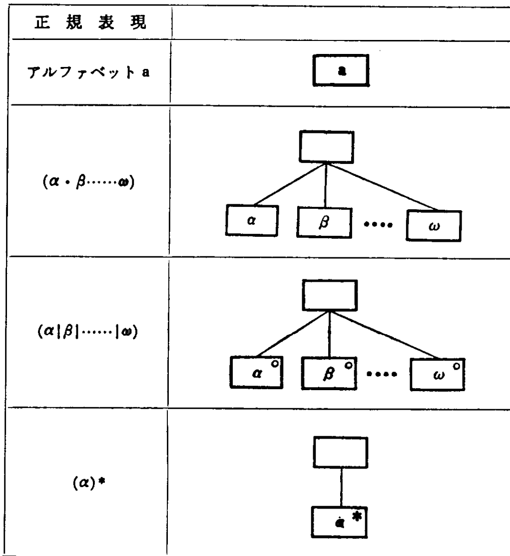


図-1 JSP 木構造図

入出力データをそれぞれジャクソンの木構造図（これを入力の木，出力の木などという）によって定義する．

2° プログラム構造の導出(プログラム・ステップ)

入力の木から出力の木への写像の存在を確かめ，入力の木と出力の木からプログラムの構造図，すなわち，プログラムの木を導出する．

3° 手続きの列挙（オペレーション・ステップ）

必要な手続き（たとえば，Cobol 文）を列挙し，プログラムの木に割り付ける．

4° プログラムの完成（テキスト・ステップ）

プログラムの実行に必要な条件を付加し，プログラム・テキストを完成する．以下，各ステップについて説明を加えよう．

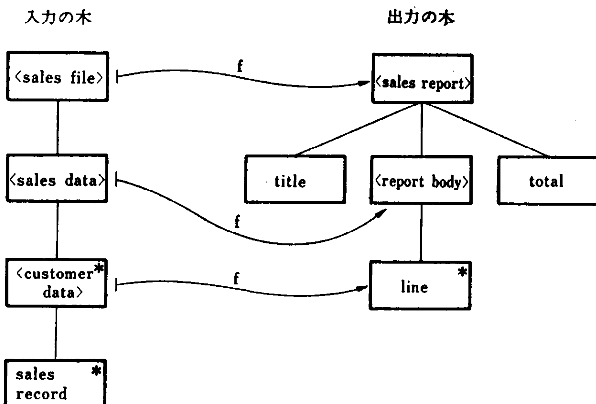


図-2 入出力の木

3.1 データ・ステップ

入力データと出力データを正規言語の文とみなすわけであるから，この第1段階の仕事は，この文法を見つけて木構造図として表現することである．正規文法の代りに正規表現を用いる．そこで正規表現を木の節に印（○と＊と無印．これを節の型とよぶことにする．）を付け，**図-1** のように表現する．これをジャクソンの木という．先の例題に適用したものが，**図-2** である（図から eof は省略する）．木の節のうち最上端の節を根，各枝の先端の節を葉とよんで区別することもある．

3.2 プログラム・ステップ

つぎのステップは，入力の木から出力の木への写像の存在を確かめることである．これは，以下の JSP が適用できるための条件になるものである．さて，この写像 f はつぎのようなものである．

定義域を入力の木の節の集合とし，値域を出力の木の節の集合とする写像 f は，

1° 木の節の行きがけ順 (preorder) を保存する．すなわち，木の節を訪問する順序で，まず木の根を訪問し，つぎに最左部分木を訪問し，続いて順次その右隣の部分木を訪問するものを行きがけ順といっているが，この順序を f は保存する．

2° 木の葉の型を保存する．ただし，その縮退型を許す（縮退型とは，原像の節のもつ＊や○印が像の節でもたないもの，また **図-3** にみられるように，同値なデータの木で単純化したもの）．

3° 入力の木の葉の f による像は定義されれば出力の木の葉である．入力の木においてある葉の像が定義されるとき，その葉から木の根にいたる分岐上の節は，すべて f による像が定義されなければならない．また出力の木においてその葉の f による原像が存在するとき，葉から根にいたる分岐上の節にすべてその原像をもたなければならない．

ここでつぎを注意する．写像の構成のために必要なら $\alpha|\beta=\beta|\alpha$, $(\alpha^*)^*=\alpha^*$, $\alpha^*\cdot\alpha^*=\alpha^*$, $\alpha\varepsilon=\varepsilon\alpha=\alpha$（ε は空記号列）などを考えて木を書き直すといい．

以上のような写像 f が存在するとき，入力の木と出力の木とは構造一致であるといい，存在しないとき，構造不一致 (structured clash) であるという．以下しばらくは構造一致の場合だけを考え

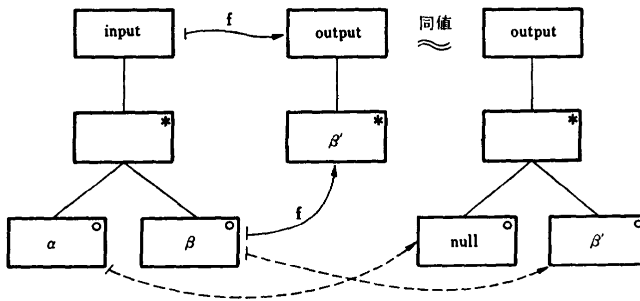


図-3 縮退型の例

る．例題の場合，図-2 の矢印 f が示すように確かに写像を構成することができる．

構造一致の場合には入力の木に出力の節を追加することでプログラム構造を導出できる．つまり，出力の木を行きがけ順に眺めてその節が入力に原像をもたないものを入力の木に補足する．こうして得られた木をプログラムの木という．先の例題でプログラムの木をつくり，節の名をプログラムらしくつけかえたものが**図-4** である．

ここでこの木に対してつぎのような解釈をするのでプログラムの木といったわけである．プログラムの木で，$\alpha \cdot \beta$ に相当するものに対しては，perform α-procedure perform β-procedure, $\alpha | \beta$ に相当するものに対しては，if condition α perform α-procedure else perform β-procedure, α^* に対しては，perform α-procedure until condition not α とみなして，木の行きがけ順にこの節の解釈を実行する．

3.3 オペレーション・ステップ

これはプログラムとして必要な手続きを列挙する段階である．手続きは使用プログラム言語によって書き下せばよい．

例題に対して Cobol によってこの段階を実行すれば，つぎのようになる．まず，出力に関する手続きを列挙する．

1. open output reportfile
2. close reportfile
3. write titlerecord
4. write customerrecord
5. write totalrecord

つぎに，出力データを導き出すための手続きを出力から入力の方向へたどる．

6. move 0 to total
7. move 0 to customersum
8. add record to customersum
9. add customersum to totalrecord
10. move customerkey to current-key

最後に，入力に関する命令を洗い出す．

11. open input salesfile
12. close salesfile
13. read salesfile

ここで手続きの列挙を，出力，演算，入力と分類して行ったのは，列挙された手続きをプログラムの木に割り付けるつぎの作業がうまく実行できないときは，プログラム構造と手続きの列挙を再点検しようと意図したからである．

まず，列挙された手続きをひとつひとつプログラムの木の適切な葉に割り付ける．もしこのとき不都合がおこれば，プログラムの構造なり手続きの列挙を疑ってみるといい．この段階は有用な検査段階であると同時に，構造の改良を行うことができる段階でもある．この割り付けは，手続きの左側の番号をプログラムの木に書き込むことで表現する (**図-5**)．

容易にわかることであるが，read 文と write 文は，それぞれ入力の葉，出力の葉から導かれたプログラムの木の葉の中にだけ現われる．ただし，Cobol を前提とすれば，read 文は先読みの原則にしたがい，open 文の直後にもでてくる．

3.4 図テキスト・ステップ

このステップは最終的なプログラム・テキストをつくる段階である．つぎに，プログラムの木に実行に必要な，選択文や反覆文に条件式を加えて，プログラム・テキストに変換する．ここで条件式が書けるこ

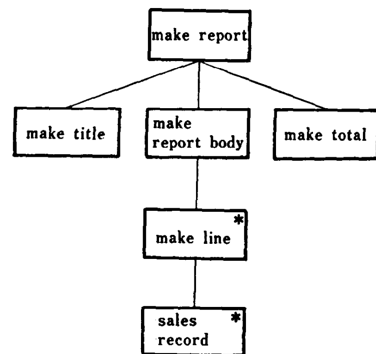


図-4 プログラムの木

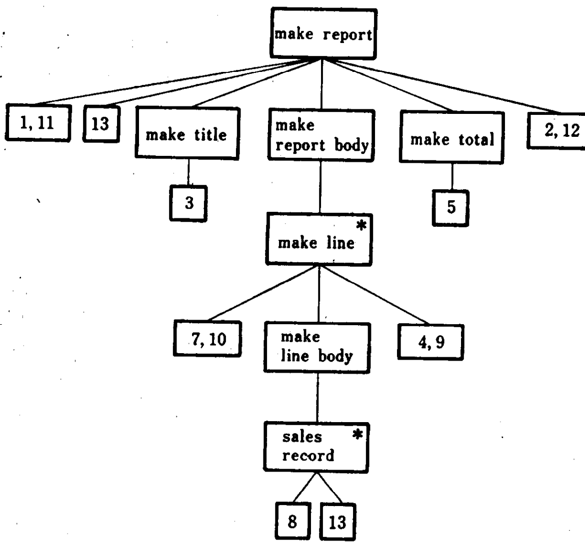


図-5 手続きの割り付け

```
MAKE-REPORT SEQ
  DO 1, 11  DO 13
  MAKE-TITLE SEQ
    DO 3
  MAKE-TITLE END
  MAKE-REPORT-BODY ITR UNTIL EOF
    MAKE-LINE SEQ
      DO 7, 10
      MAKE-LINE-BODY ITR WHILE (CUSTOMKEY = CURRENTKEY)
        SALES-RECORD SEQ
          DO 8  DO 13
        SALES-RECORD END
      MAKE-LINE-BODY END
      DO 4, 9
    MAKE-LINE END
  MAKE-REPORT-BODY END
  MAKE-TOTAL SEQ
    DO 5
  MAKE-TOTAL END
  DO 2, 12
MAKE-REPORT END
```

図-6 図式理論

と，すなわち，その条件式がそこで評価できることが大切である．

ジャクソンは，図式論理 (schematic logic) という独自の表現法を考案し，図式論理による記述を Cobol コードに変換する Cobol 前処理系 JSP-COBOL を用意している．これについてはあとで触れる．例題を図式論理で表現したものが **図-6** である．図中の符号は以下のように解釈する．

seq……end は，それらに挟まれた手続きを順次に実行する．itr until (反覆終了条件) ……end, itr while (反覆条件)……end, これらはともに手続きを繰り返し実行する．このほかに，sel (選択条件 1)……alt (選択条件 2)……end がある．これは選択条件に応じていずれかの手続きを実行することを意味する．

以上みてきたのが基本ジャクソン法である．

4. 条件判定困難な課題

木の節の型のうち，∘印のものを選択，＊印のものを反覆とよぶことにすると，基本 JSP のテキスト・ステップで選択と反覆の節に条件を書き加えなければならないことはすでにみてきたとおりである．場合によってはこの条件式の評価がその必要なときにできない事態も起り得る．たとえば，選択枝を実行してはじめて条件式が評価できるような場合である．先の例では，1 件のレコードを先読みすることによってすべての反覆条件を評価することができた．場合によっては，複数件のレコードを先読みするとよいことがある．この条件式を評価できない場合をジャクソンは条件判定困難な課題 (recognition difficulty) とよんでいる．たとえば，固定数のデータと 1 件のチェックサム・データからなるデータ・ブロックを読み込んで正しいデータであるか否かを調べるためには，1 ブロック分のデータをいったん主記憶装置にすべて読み込まなければ正しいデータであるか否かを評価できない．

条件判定困難な課題に対する解決策としてジャクソンは，多段先読み (multiple read ahead) と後戻り (backtracking) を紹介している．多段先読みは，固定数のレコードを先読みすれば条件を評価できる課題 (たとえば，チェックサムの問題) に向いている．入力ファイルの open 文に続いて条件を評価できるだけの固定複数回の連続した read 文を出し，以降の read 文は必要な個所で 1 回ずつ出せばよい．この方法は，先につくられているプログラムの木を手直ししないでプログラムの字面を修正するだけですむ．可変数のレコードを先読みしなければ条件を評価できない課題に対しては，一般に後戻りの方法が適している．

ジャクソンは後戻りをつぎの 3 つのサブステップに手順化している．

段階 1；条件仮定ステップ

ある条件が正しいと仮定する．選択の節では一方が正しいと仮定する．反覆の節では反覆がまだ続くと仮定する．

段階 2；仮定検定ステップ

段階１で立てた仮定のもとで実行を行い，仮定した条件を調べて仮定が間違っていることが発見されれば，その時点で仮定を棄却する．

段階３；副作用の補修ステップ

副作用とは，仮定外で加工されるべきデータに関する，仮定内での加工の結果である．仮定外で生かせる有益な副作用か，修復しなければならない有害な副作用かを見い出し，仮定棄却後の仮定外でそれらに応じた処置を行う．

後戻りと副作用を理解するために，以下の小さな課題を考える．

〈索表問題〉「参照キーをもったデータの一次元配列として構成された表を，検索キーをもとに配列順に索表する．検索キーと同じキーをもったデータがあるかないかを調べ，最初にみつかったデータを選び出す.」

この表の木は **図-7** のようになる．プログラムの木は，この表の木から導かれる．段階１では，表に検出すべきデータが存在しないものと仮定する．プログラムは「検索レコードがない表」の節の実行を開始する．段階２では，この「検索レコードがない表」の節を実行しながら検索レコードが存在しないという仮定を調べる．もし，実行途中で捜しているレコードが発見されれば，ただちに実行を「検索レコードがある表」の節に移す．段階３では，「検索レコードより前のレコード群」の節を実行する必要はないことを確認する．なぜならば，前の「検索レコードがない表」の節で同じ手続きをすでに実行している（仮定外にとって有益な副作用である）からである．この例では有害な副作用はないから修復作業は必要としない．プログラムの木をつくり，JSP-COBOL 流に書いたプログラムは **図-8** になる．

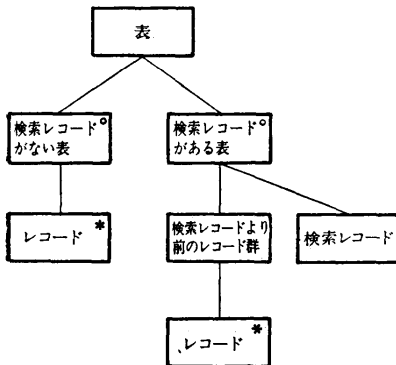
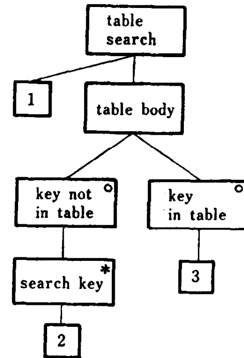


図-7　表の木




```
1. i←1
2. i←i+1
3. 索表後の実行
  tablesearch seq
    do 1
    tablebody posit (key not in table) …仮定設定
    tablebody quit (key=table (i))…仮定検定
      do 2     (注．（ ）内が真になると制御は admit へとぶ)
    tablebody admit (key in table)…仮定棄却
      do 3
    tablebody end
  tablesearch end
```

図-8　表のプログラムの木とプログラム

後戻りは多段先読みと同じく，プログラムの木を手直しすることなく課題をプログラミングすることができる．後戻りに適切な機能をもたない言語では go to 文を使用することになるが，目角を立てるほどのことでもあるまい．

5. 構造の不一致

今までの例は，入力と出力のデータ構造間に写像を見い出してプログラム構造をつくることができる構造一致の場合にかぎられていた．（こうしてつくられたプログラムを「単純な」プログラムとよぶことにする.）つぎに構造不一致の場合をみてみよう．構造不一致として３つの例をあげる．つまり，順序不一致，境界不一致，入り交じり不一致である．

1°　順序不一致 (ordering clash)

入力木の節の原像とそれが結ぶ出力木の節の像とを行きがけ順で比較したとき，入力データ列と出力データ列の順序が異なることがある．この場合には基本 JSP でプログラムをつくることができないので，つぎの手順をふんで解決する．入力データ列を出力データ列に変換する場所に十分な記憶場所を設け，そこに出力データ列と同じ順序に入力データ列を並びかえてたくわえる．こうすれば，プログラムはデータを記憶場所に書き込むものとそこから読み出すものと２つの

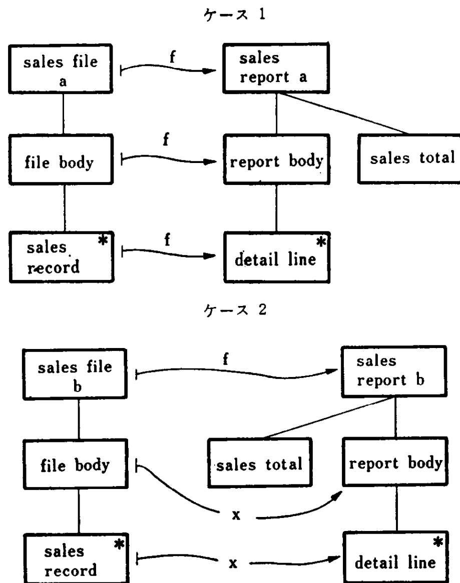


図-9 順序不一致の例

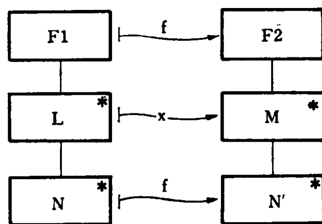


図-10 境界不一致

プログラムに分割される．後者は，単純なプログラムとして基本 JSP でプログラミングできる．前者は，整列プログラムを使うか，記憶場所に内部表を使うか等は課題に応じて決めればよい．

ここで簡単な例題を考える．「複数の売上レコードからなる売上ファイルを入力し，その売上明細と売上合計行を印書する．このとき売上合計行を最初に印書する場合と，最後に印書する場合について考える．」

基本 JSP のプログラム・ステップで入力の木，出力の木を行きがけ順に眺めて，プログラムの木を導出できるかどうかを確認する．図-9 においてケース2

図-11 境界不一致の解法

顧客	売上	ページ1
A	100	
A	200	
A TOTAL	300 *	
B	10	
B	20	
顧客	売上	ページ2
B	50	
B TOTAL	80 *	

(顧客A / 顧客B / ページ1 / ページ2)

図-12 売上集計表

の場合は sales file b と sales report b の間で sales total と detail line を同時に導くことができない，つまり，1本のプログラムの木を導出できないことがわかる．この例は順序不一致として同様の方法でプログラミングできる．

2° 境界不一致 (boundary clash)

入力の木と出力の木の間で，上位の節間に写像があり，下位の節間でも同様に写像があるが，中間の節間に写像が見い出せない場合 (図-10)，この2つのデータ間には境界不一致があるという．

この場合，基本 JSP ではプログラムをつくることができないので，2つのプログラムに分割して解決する．中間に順編成ファイル (図-11 の FX) を導入し，FX ファイルにデータを書き込むプログラム P1 と FX からデータを読み込むプログラム P2 とに分割する．以下の事例で解法を説明する．はじめにみた簡単な事例を以下のように変更する．

「顧客ごとに整列されている売上データを入力し，顧客ごとに売上明細と合計行を印書する．集計表には，ページごとに頭書きを印書する．」(図-12)

この集計表に関して，〈page—detail line〉という物理的構造と〈customer—sales record〉という論理的構造を同時に含んだデータの木を描くことはできない．そのためつぎのように考える．集計表をページという物理的概念からみたデータの木として把え (図-13 の report file)，その木に対して，原像を見い出し得る中間ファイル inter file を導入する．この inter file は売上データ・ファイルに顧客ごとの売上合計を追加したものに等しい．プログラムを，sales file から inter file に変換するプログラム P1 と inter file を report file に変換するプログラム P2 とに分割し，この inter file を2つに解釈する．

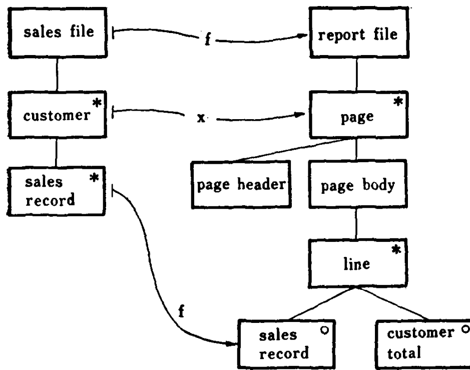


図-13 データの木 (境界不一致)

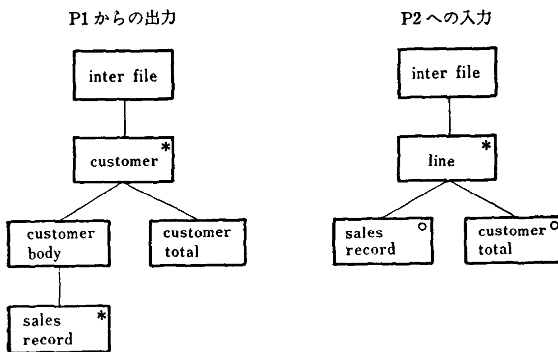


図-14 中間ファイルの木

ひとつは, sales file の像をもつものとして, 他方は, sales report の原像をもつものとして解釈する (図-14). プログラム P1 と P2 が図-14の木から導かれることは容易に理解できる. 実際に物理的な中間ファイルを導入することは処理効率を悪くするが, コルーチンを用いて, 物理的ファイルを媒介することなく2つのプログラムを結合すれば, その点は改善できる*.

ジャクソンはこれをプログラム・インバージョン (プログラムの転換) とよぶ技法で実現している. この技法については次章にゆずる.

3° 入り交じり不一致 (interleaving clash)

以下の例題を考える. 「入力ファイル userlog にはシステム稼動中に記録された端末ごとの開始時刻, 終了時刻が発生時刻順に記録されている. 出力ファイル userrpt には端末ごとの使用時間 (終了時刻—開始時刻) を記録する. 端末は複数個あるが, ひとつの端末は1回しか使用しないものとする. 出力データの順序は問わない.」

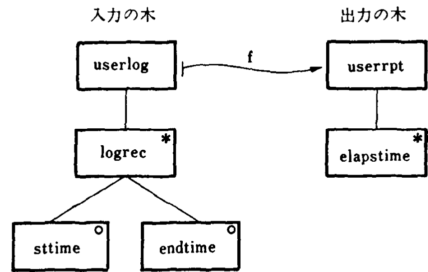


図-15 入出力の木 (入り交じり不一致)

この入出力データの木は図-15になる. elapstime を導き出すデータ sttime と endtime は, 入力の木の中に存在するが, 端末ごとの sttime と endtime の順序が保障されているだけで端末間のデータには一定の関係がない. こうした場合, 入力と出力の間には入り交じり不一致があるという. 入り交じり不一致を解決する方法は3つある.

第一は, 順序不一致の解法と同じように, 中間に物理的ファイルを設け, userrpt のデータ列と同じ順序に整列した中間ファイルを生成するものである. この場合, 中間ファイルを完全に生成するまで出力できない.

第二は, 端末の数だけ中間ファイルとプログラムを生成するものである (図-16). この方法もすべての入力データを split が処理し終るまで出力できないという制約がある.

第三は, 図-16の中間ファイル F1,〜, F_n をインバージョンする方法である. userrpt を生成するプログラムは1本となり, P1,〜, P_n が平行的に実行されるのと同じ効果をもつ. 入力 userlog を完全に入力し終る前に出力 userrpt を出力し始めることができる*.

6. 郵送ラベル作成問題

プログラム・インバージョンを理解するために, つぎの課題をプログラミングする.

「郵送ファイルから郵送ラベルを作成する. 郵送ラ

* この例題に限っていえば, Cobol の報告書機能を使うなり, プログラム構造にページの概念を含めない, すなわち, 印書手続きにページの制御を含めてしまうなどの工夫をすれば, わざわざコルーチン化を試みる必要もないわけである.

* M. ジャクソンは, JSP をシステム開発技法 JSD (Jackson System Development) に発展させている[5]. この中で入り交じり不一致の概念は大きな比重を占めているが, ここでは詳細な記述を省略する. なお, JSD は近く書籍として出版される予定と聞く.

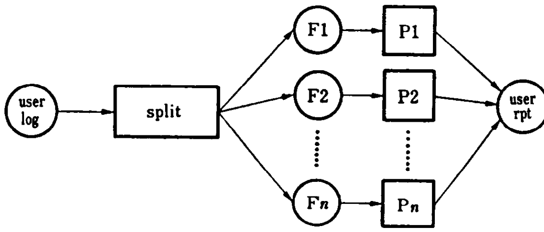


図-16　入り交じり不一致の解法

ベルの形式は，各宛先ごとに名前が1行あり，つぎに0を含む複数件の住所行が続き，最後に1行の空白行からなる．郵送ファイルは，宛先ごとに整列され，連続した語からなるカードの集まりである．入力の各宛先は，名前を構成する語の集まりと住所を構成する語の集まり（0件のこともある）からなっている．郵送ファイルの語は，1個以上の空白文字によって分離されている．語は1枚のカード・レコードに完全に含まれている．名前の最初の語は，先頭に斜線 (/) が付く．各々の住所行の最初の語には，先頭にハイフン (-) が付く．これらの識別文字は出力から取り除かれるが，語の中に現われる斜線とハイフンは，識別文字ではないので取り除かれない．郵送ファイルはつぎのことが保証されている．すなわち，各宛先は6行以上含まないということと，印刷用に名前行と住所行から識別用の斜線とハイフンを取り除いて，語と語の間の空白を1個に減らすと，それらの1行は 32 文字以上は含まないということである．斜線またはハイフンだけの語もないということも保証されている．以下に，実際のデータ例を示す．

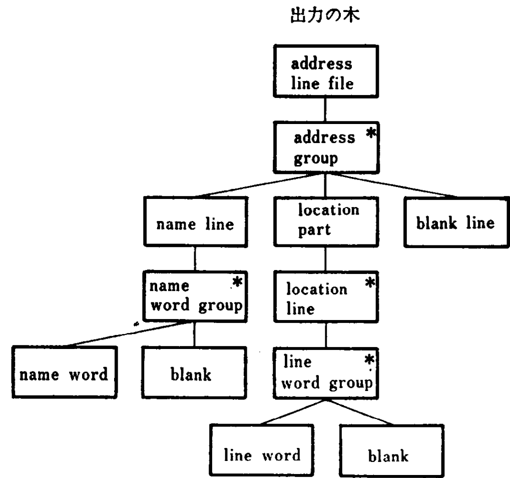
/JOHN SMITH　-5 SEAVIEW ROAD
　-BRIGHTON
　-SUSSEX /JOAN　WILLIAMS -FLAT 4
　-THE PROMENADE　-BLACKPOOL

出力はつぎのようになる．

JOHN SMITH
5 SEAVIEV ROAD
BRIGHTON
SUSSEX

JOAN WILLIAMS
FLAT 4
THE PROMENADE
BLACKPOOL

出力の木



入力の木

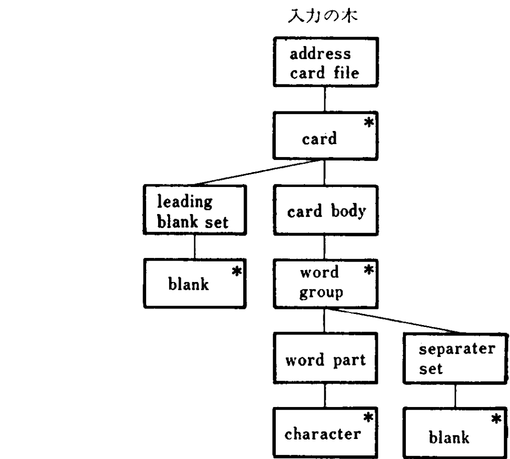


図-17　データ構造図

入力ファイルと出力ファイルのデータ構造は 図-17 のようになる．理解を容易にするために，データ構造から blank と character の節を取り除き，name line と location line の区別を失くすと，図-18 のようなスケルトンができあがる．上位の節 address card file と address line file 間に，写像があり，下位の word 間にも写像がみられるが，中の節には写像が見い出せない．したがって，2つのデータ構造間には境界不一致が存在することがわかる．

この境界不一致を解決するために，中間にファイルを設け，入力ファイルを中間ファイルに変換するプログラム P1 と中間ファイルを出力ファイルに変換するプログラム P2 に分割する．中間ファイル inter file を構成するレコードは，下位の像をもつ節 word が適切である．P1 からみた inter fileは，単に word の

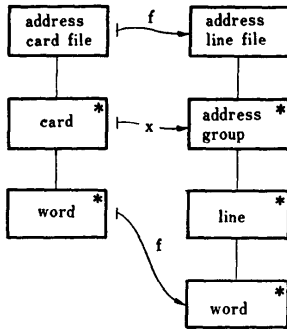


図-18 データ構造のスケルトン

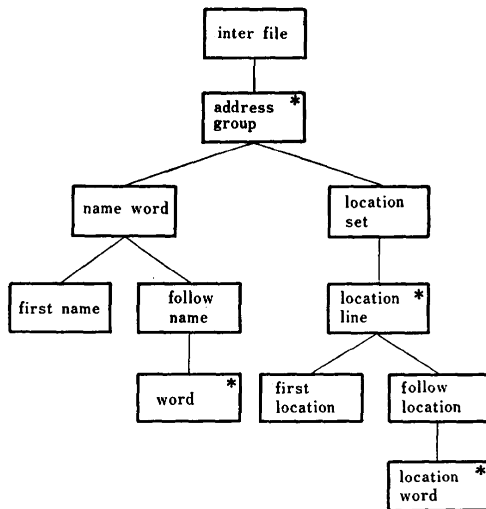
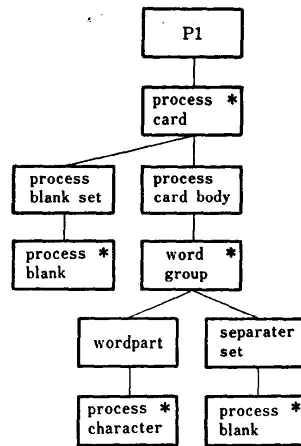


図-19 P2 側の中間ファイル



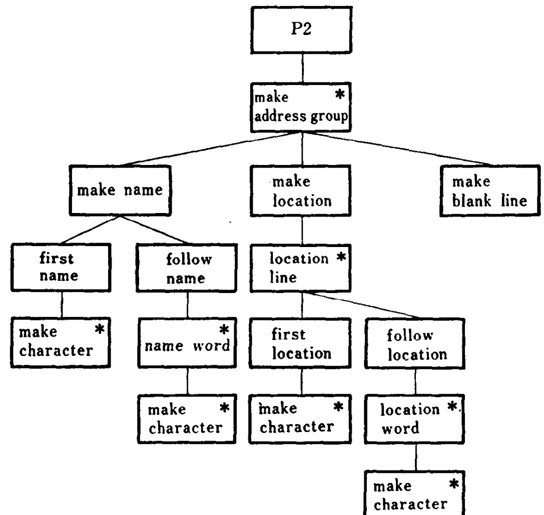


図-20 P1, P2 プログラムの木

繰り返しである. P2 からみた inter file は 図-19 のようになる. これらをもとに P1 と P2 のプログラムの木を導くと 図-20 が得られる. つぎに必要な命令を洗い出し, プログラムの木に割り付ける. さらに図式論理でプログラム・テキストに変換し, 前処理系 JSP-COBOL に入力し, 得られた原始コードが 図-21 である. このコードは通常の Cobol 処理系への入力となる*.

JSP-COBOL ではコルーチンの発想を取り入れた機能をもっているので, P1 と P2 を独立に作成し, P1 では word を interfile に write し, P2 では inter file から word を read するかのようにプログラムを書いてよい. 実際には, 物理的な inter file はつくられることなく, word 1 件ごとに P1 と P2 の間でデータと制御の移動が行われる. すなわち, P1 で word を write すると, 制御は, P2 内でその word を read している個所に移る. P2 で word の処理が完了し, つぎの word を read する直前に制御は P1 の先ほどの write 文のつぎの命令に移動する. コルーチンもしくは類似した機能をもたない言語ではインバージョン・コーディングを行うことによって, プログラムの構造を変えることなくプログラムを書き直すことができる. 図-21 のプログラムを, P1

* この処理系の日本における販売代理権は(株)日本ビジネス・オートメーション (JBA) が保有しており, 日本ユニバック(株)では教育と紹介を行っている.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADDRESS-PRINT.
AUTHOR.
    EDUCATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    UNIVAC-1100.
OBJECT-COMPUTER.
    UNIVAC-1100.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ADDRESSCARDFILE ASSIGN TO CARD-READER.
    SELECT INTERFILE ASSIGN TO SI-S-P2. <----------中間ファイルを定義している
    SELECT ADDRESSLINEFILE ASSIGN TO PRINTER.
IMPLEMENTATION SECTION.
MI  ADDRESS-PRINT.
    MAKE P1 CONTROLLER. <----------主プログラムを指定している
DATA DIVISION.
FILE SECTION.
FD  ADDRESSCARDFILE LABEL RECORD OMITTED.
01  CARD-REC.
 02 CARDCOLUMN                          PIC X            OCCURS 80 TIMES.
FD  ADDRESSLINEFILE LABEL RECORD OMITTED.
01  ADDRESSLINE-REC                     PIC X(132).
FD  INTERFILE LABEL RECORD OMITTED.
01  INTERFILE-REC                       PIC X(34).
01  INTER-RECORD.
 02 INTER-COLUMN                        PIC X            OCCURS 34 TIMES.
WORKING-STORAGE SECTION.
01  WORDRECORD.
 02 WORDCHAR                            PIC X            OCCURS 34 TIMES.
01  LOCATION-REC.
 02 LOCATION-COLUMN                     PIC X            OCCURS 34 TIMES.
01  I                                   PIC 9(2).
01  J                                   PIC 9(2).
01  L                                   PIC 9(2).
01  M                                   PIC 9(2).
PROCEDURE DIVISION.
PRGM P1.
OPERATIONS.
1.    SOPEN OUTPUT INTERFILE.
2.    SCLOSE OUTPUT INTERFILE.
3.    SWRITE INTERFILE-REC FROM WORDRECORD.
4.    MOVE SPACES TO WORDRECORD.
5.    MOVE 1 TO I.
6.    ADD 1 TO I.
7.    MOVE 1 TO J.
8.    ADD 1 TO J.
9.    MOVE CARDCOLUMN (J) TO WORDCHAR (I).
10.   SOPEN INPUT ADDRESSCARDFILE.
11.   SCLOSE INPUT ADDRESSCARDFILE.
12.   SREAD ADDRESSCARDFILE.
```

図-21 ①

を主プログラム，P2 をサブプログラムとして，Cobol でインバージョンを行う例を図-21 に示しておいた．サブプログラムに制御を受けたときの再開番地を示す状態変数 QS を導入することによって，いささかもプログラムの構造を変更することなく，インバージョンできることがわかる．

7. おわりに

JSP は2つの特長に要約できる．

第1は，JSP は事務計算分野に応用度の高いプログラミング方法論であるという点である．JSP では，プログラムの入出力データにひそむ正規言語の構造を見い出し，その構造からプログラムを誘導する．順編成ファイルはもちろんのこと，データベースであろうと，会話型端末からの入出力であろうと，そこに正規文法をみいだし，プログラムを作成する．ファイルの変換過程として実現される事務計算分野には自然応用度が高い．

第2の特長は，JSP は課題志向型方法論 (problem oriented method) であるということである．JSP では，課題の認識をデータの定義を通して行う．得られたデータの定義からプログラムを導くために課題に応じたプログラムをつくることができる．他のよくあるプログラミング方法論，たとえば，構造化設計[6]は，最初に機能を分析することによってプログラムの構造をつくろうとする．機能の分解の仕方が作成する人によって異なるため，いくつかのプログラム構造がつくられ得る．いわば解法志向型方法論 (solution orien-

```
STRUCTURE.
P1  SEQ
  DO 1     SOPEN OUTPUT INTERFILE.
     10    SOPEN INPUT ADDRESSCARDFILE.
  DO 12    SREAD ADDRESSCARDFILE.
     7     MOVE 1 TO J.
  P1BODY ITR WHILE (ADDRESSCARDFILE-OK)
    PROCESS-CARD SEQ
      PROCESS-BLANKSET ITR WHILE (CARDCOLUMN (J) EQ SPACE)
        DO 8     ADD 1 TO J.
      PROCESS-BLANKSET END
      CARDBODY ITR WHILE (J LE 80)
        WORDGROUP SEQ
          DO 4     MOVE SPACES TO WORDRECORD.
             5     MOVE 1 TO I.
          WORDPART ITR WHILE (CARDCOLUMN (J) NE SPACE)
            DO 9     MOVE CARDCOLUMN (J) TO WORDCHAR (I).
            DO 6     ADD 1 TO I.
               8     ADD 1 TO J.
          WORDPART END
          SEPARATERSET ITR WHILE (CARDCOLUMN (J) EQ SPACE AND J LE 80)
            DO 8     ADD 1 TO J.
          SEPARATERSET END
          DO 3     SWRITE INTERFILE-REC FROM WORDRECORD.
        WORDGROUP END
      CARDBODY END
      DO 12    SREAD ADDRESSCARDFILE.
         7     MOVE 1 TO J.
    PROCESS-CARD END
  P1BODY END
  DO 2     SCLOSE OUTPUT INTERFILE.
     11    SCLOSE INPUT ADDRESSCARDFILE.
P1  END
PRGM P2.
OPERATIONS.
1.    SOPEN OUTPUT ADDRESSLINEFILE.
2.    SCLOSE OUTPUT ADDRESSLINEFILE.
3.    SWRITE ADDRESSLINE-REC FROM LOCATION-REC.
4.    MOVE INTER-COLUMN (M) TO LOCATION-COLUMN (L).
5.    MOVE 1 TO M.
6.    MOVE 1 TO L.
7.    ADD 1 TO M.
8.    ADD 1 TO L.
9.    MOVE SPACES TO LOCATION-REC.
10.   SOPEN INPUT INTERFILE.
11.   SCLOSE INPUT INTERFILE.
12.   SREAD INTERFILE.
STRUCTURE.
P2  SEQ
  DO 1     SOPEN OUTPUT ADDRESSLINEFILE.
     10    SOPEN INPUT INTERFILE.
  DO 12    SREAD INTERFILE.
  P2BODY ITR WHILE (INTERFILE-OK)
```


図-21 ②

ted method) といってよい. JSP は, そうした方法とは決定的に異なっている[7)-9)].

最後に, 本稿執筆にあたって貴重なコメントをいただいた数人の会社の同僚に感謝する.

参 考 文 献

1) Jackson, M. A.: Principles of Program Design, pp. 299, AcademicPress, London (1975). (邦訳 鳥居宏次: 構造的プログラム設計の原理, 日本コンピュータ協会 (1980)).

2) Hughes, J. W.: A Formalization and Explication of the Michael Jackson Method of Program Design, Software-Practice and Experience, Vol. 9, pp. 191-202 (1979).

3) 中田育男: コンパイラ, pp. 278, 産業図書, 東京 (1981).

4) 福村晃夫・稲垣康善: オートマトン・形式言語理論と計算論, pp. 235, 岩波書店, 東京 (1982).

5) Jackson, M. A.: Information Systems: Modelling, Sequencing and Transformation, Proc. 3rd ICSE, pp. 72-81 (1978).

6) Yourdon, E./Constantine, L. L.: Structured Design, pp. 473, Prentice Hall, New Jersey (1975).

7) Molluzzo, J. C: Jackson Techniques for Elementary Data Processing, Bull. SIGCSE, Vol. 13, No. 4, pp. 16-20 (1981).

8) Bergland, G. D: Structured Design Methodologies, Proc. 15th Design Automation Con-

```
ADDRESSGROUP SEQ
  MAKE-NAME SEQ
    DO 5     MOVE 1 TO M.
       6     MOVE 1 TO L.
       9     MOVE SPACES TO LOCATION-REC.
    FIRSTNAME SEQ
      DO 7     ADD 1 TO M.
      FIRSTNAMEBDY ITR WHILE (INTER-COLUMN (M) NE SPACE)
        DO 4     MOVE INTER-COLUMN (M) TO LOCATION-COLUMN (L).
        DO 7     ADD 1 TO M.
           8     ADD 1 TO L.
      FIRSTNAMEBDY END                               [MOVE 3 TO QS.  EXIT.
      DO 8     ADD 1 TO L.                            P2-03.]
    FIRSTNAME END
    DO 12  [SREAD INTERFILE.] <-
    FOLLOWNAME ITR UNTIL (INTERFILE-EOF OR INTER-COLUMN (1) EQ '/' OR INTER-COLUMN (1) EQ '-')
      NAMEWORD SEQ
        DO 5     MOVE 1 TO M.
        NAMEWORDBDY ITR WHILE (INTER-COLUMN (M) NE SPACE)
          DO 4     MOVE INTER-COLUMN (M) TO LOCATION-COLUMN (L).
          DO 7     ADD 1 TO M.
             8     ADD 1 TO L.
        NAMEWORDBDY END
        DO 8     ADD 1 TO L.                          [MOVE 4 TO QS.  EXIT.
        DO 12  [SREAD INTERFILE.] <-------            P2-04.]
      NAMEWORD END
    FOLLOWNAME END
    DO 3     SWRITE ADDRESSLINE-REC FROM LOCATION-REC.
  MAKE-NAME END
  MAKE-LOCATION ITR WHILE (INTERFILE-OK AND INTER-COLUMN (1) NE '/')
    LOCATION-LINE SEQ
      DO 6     MOVE 1 TO L.
         9     MOVE SPACES TO LOCATION-REC.
      FIRSTLOCATION SEQ
        DO 5     MOVE 1 TO M.
        DO 7     ADD 1 TO M.
        FIRSTBDY ITR WHILE (INTER-COLUMN (M) NE SPACE)
          DO 4     MOVE INTER-COLUMN (M) TO LOCATION-COLUMN (L).
          DO 7     ADD 1 TO M.
             8     ADD 1 TO L.
        FIRSTBDY END                                  [MOVE 5 TO QS.  EXIT.
        DO 8     ADD 1 TO L.                           P2-05.]
        DO 12  [SREAD INTERFILE.] <-
      FIRSTLOCATION END
      FOLLOWLOCATION ITR WHILE (INTERFILE-OK AND INTER-COLUMN (1) NE '/' AND INTER-COLUMN (1) NE '-')
        LOCATIONWORD SEQ
          DO 5     MOVE 1 TO M.
          LOCATIONBDY ITR WHILE (INTER-COLUMN (M) NE SPACE)
            DO 4     MOVE INTER-COLUMN (M) TO LOCATION-COLUMN (L).
            DO 7     ADD 1 TO M.
               8     ADD 1 TO L.
          LOCATIONBDY END
          DO 8     ADD 1 TO L.


          DO 12  [SREAD INTERFILE.] <-------          [MOVE 6 TO QS.  EXIT.
        LOCATIONWORD END                               P2-06.]
      FOLLOWLOCATION END
      DO 3     SWRITE ADDRESSLINE-REC FROM LOCATION-REC.
    LOCATION-LINE END
  MAKE-LOCATION END
  MAKE-BLANKLINE SEQ
    DO 9     MOVE SPACES TO LOCATION-REC.
       3     SWRITE ADDRESSLINE-REC FROM LOCATION-REC.
  MAKE-BLANKLINE END
ADDRESSGROUP END
P2BODY END
DO 2     SCLOSE OUTPUT ADDRESSLINEFILE.
   11  [SCLOSE INPUT INTERFILE.] <-----------         [MOVE 1 TO QS.  EXIT.]
P2  END
```

図-21 ③

図-21 JSP-COBOL プログラム

ference, pp. 475-493 (1976). (邦訳 妹尾 稔: 構造化設計法, bit, Vol. 14, No. 3, pp. 15-40 (1982)).

9) 米口 肇: ジャクソン法, bit, Vol. 12, No. 12, pp. 61-72 (1980).

(昭和 57 年 8 月 3 日受付)