ソフトウェア工学 46－17
（１９８６．２．７）

データ構造に基づくテスト・ケース設計法

峰尾　欽二
（日本ユニバック（株）　教育部）

１．はじめに

プログラムの生産現場において，いかにテスト・データを準備するかということは常にプログラマを悩ませる問題である。プログラム・テストのためのデータ例の選択は確かに難しい問題である。計算対象が複雑な構造をもっているとその選択は一層困難になる。事務計算の対象であるファイルに対してもこの選択はやさしくない。この小論は，事務計算プログラムのテスト・データ選択についての一方法を提示する。この方法は課題から直接にデータ例を選択するもので、コードに基づいたものではない。このデータ例によってコード・カバレッジを検査し，そこで良い結果が得られればコード自身も適切に記述されているとみなしていいだろうといえるようなものである。

入力データ領域Ｉを出力データ領域Ｏに変換するプログラムＰを考える。データ例選択のためにＩを適切な基準で類別したい。そのためにはＩに属するレコードをアルファベットとする句構造言語，とくに正規言語とみなすと都合がいい。このとき，プログラムＰはその構文的側面を抽象化してＩからＯへの変換系となる。この点に着目してデータ例をつくろうというわけである。以下に例示する。

２．例１；テキスト編集問題

やや現実的でない点もあるが，入力テキストを出力テキストに変換する小さなプログラムを考える。

入力テキストの文章は，単語に続くピリオドと空白文字で区切られる。出力テキストでは，文章中に出現する１個以上の連続した空白文字は１個の空白文字に変換される。ピリオドに続く連続した空白文字は改行文字に変換される。入力テキストの最初に空白があってもそれは出力されないものとする。

この課題の入力領域に正規文法を見出して正規言語ＩＬとして修正ＢＮＦで書くと以下のようになる。

ＩＬ
```
<input-text>::= <space-seq><sentence>* end-of-text
<sentence>::= <word><word-set>* period space <space-seq>
<word-set>::= space <space-seq><word>
<word>::= not-space-character not-space-character*
<space-seq>::= space*
```

同様に出力領域を正規言語ＯＬとして修正ＢＮＦで書く。

ＯＬ
```
<output-text>::= <sentence>* end-of-text
<sentence>::= <word><word-set>* period new-line-char
<word-set>::= space <word>
<word>::= not-space-character not-space-character*
```

この課題は，入力ファイルinput-textから出力ファイルoutput-text への変換系として定義される。出力の終端記号を生成する入力側の記号列を対応させると以下のようになる。

period new-line-char ⟵ period space space*

space ⟵ space space*

not-space-character ⟵ not-space-character

end-of-text ⟵ end-of-text

<> <sentence>* ⟵ space* <sentence>*

ここで変換系として定義された課題を，入力データを出力データに変換する状態推移図で表わす。図-1において，εは空データを，スラッシュ（／）の左側は入力終端記号を，右側は出力終端記号を表わす。入力側に対応した出力が存在しない場合はスラッシュと出力終端記号を省略し，同様にε／εはεに省略している。

（正規言語として表現された入力領域と出力領域から入出力状態推移図に変換するプロセスは次章で詳しく触れる。）

課題を適切に反映したプログラムは，入出力状態推移図が示す入力データと出力データの関係を満たしている。したがって，コード化されたプログラムが課題を満たしていることを検証するために入出力状態推移図から適切なデータ例を抽出してテストすることは効果的である。プログラムが同じ振舞をすると期待されるデータ群を一つのテスト・ケースと考え，いかにして入力領域から適切なテスト・ケースを選定するかがここでの課題である。

テスト・ケースを開始ノードs から終了ノード fに至るパスとして選定する。このとき，すくなくともすべてのアークは網羅されなければならない。

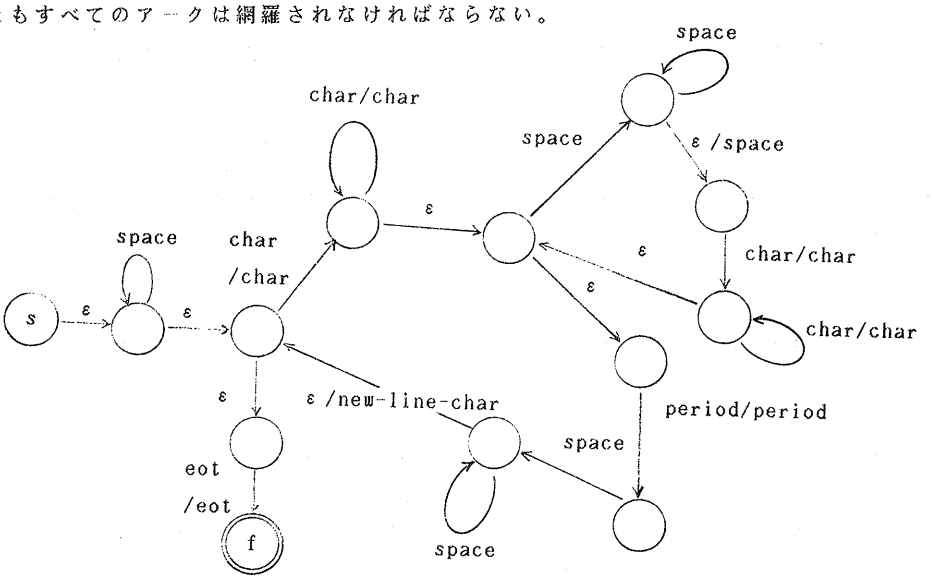


図-1　入出力状態推移図

s からf に至る，最短距離ですべてのアークを網羅するパスとしてテスト・ケース

space char char space space char char period space eot

が選定される。このとき出力されるデータは、

char char space char char period new-line-char eot

である。以上が基本的なアイデアである。

3．テスト・ケース選定法

以上みてきたアイデアを以下で詳細に述べる。
選定法は3っのステップから成り立っている。

（1）データ構造の記述

ここでは主としてＪＳＰ(Jackson Structured Programming)流で書かれたコードをテストの対象と考えているので，最初に，ＪＳＰの流儀にしたがって，入出力データの構造を木構造図で描く。入力ファイルが複数本ある場合には1本の論理的入力ファイルに統合する。（論理的入力ファイルとは，複数の入力ファイルを1本に組み合わされたものとみなしたファイルのことである）

（2）データ構造図から状態推移図への変換

まず，（論理的）入力ファイルと出力ファイルたちからＪＳＰ流のプログラム構造図をつくる。つぎに，テスト・ケースの選択を容易にするためにそのプログラム構造図を入出力状態推移図に変換する

（3）テスト・ケースの選択

入出力状態推移図からパス・カバレッジを基本にテスト・ケースを選択する。

ここではデータ及びプログラムの構造の表記は，ＪＳＰの木構造図のそれにしたがう。木構造の要素と状態推移図との対応を以下に示す。補足的にＢＮＦの表記も添書きする。

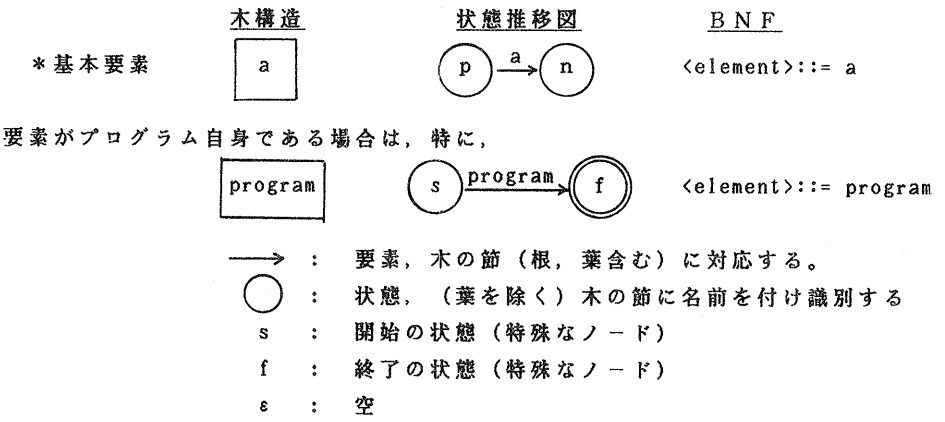


図-2　木構造と状態推移図－1

図-2は，プログラムの要素aによって，状態pが状態nに変化することを示している。

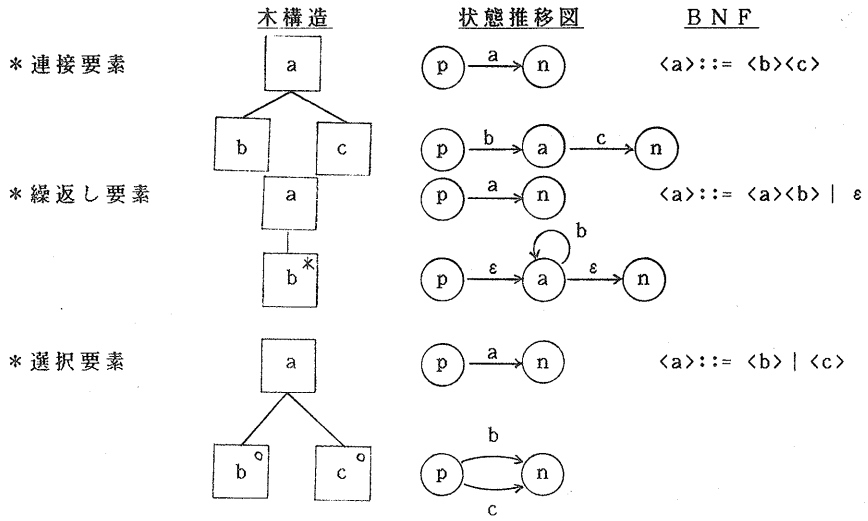


図-3　木構造と状態推移図-2

これら表記法を用いれば，いかなるＪＳＰ流の木構造図も状態推移図に変換することができる一例を図-4に示す。

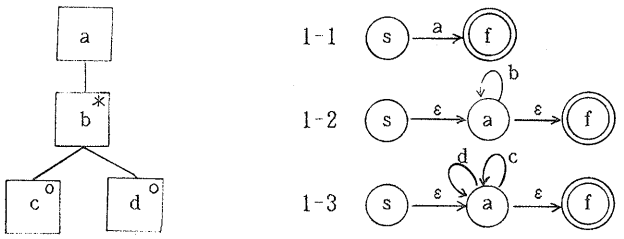


図-4　状態推移図の段階的作成

４．例２；売上表作成問題

当期の売上表を作成する。入力は製品コードをキーとして整列した売上レコードのファイルである．この売上ファイルは，当期の売上レコードと旧期の売上レコードを含んでいる。当期の売上レコードに関する売上表を作成する。最初に製品ごとの売上げ高明細を印書し，続いて製品売上げ高合計を印書する。最後に当期の総売上げ高合計を印書する。

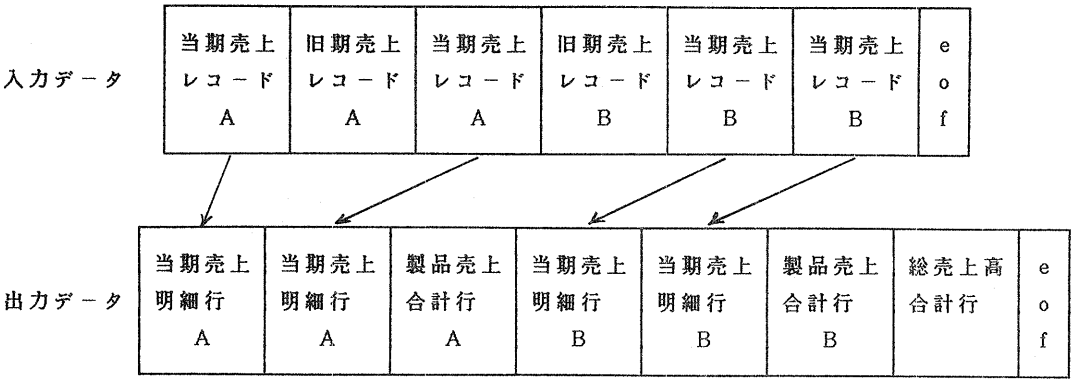


図-5　データ例

最初に，入出力データの構造図をつくる。（図-6）　つぎに，入力の木を出力の木に変換するプログラムの木をつくる。（図-7）　図-7において，四角で示されているプログラムの木の節のなかの名称ａ／ｂは，入力ａに対してｂが出力するプログラムのプロセスを表わしている。製品(*)/表本体は，多数の製品が入力した時点で表本体が出力することを，ε/総合計は，それ以前のプロセスが完了した後に総合計が出力することを示している。

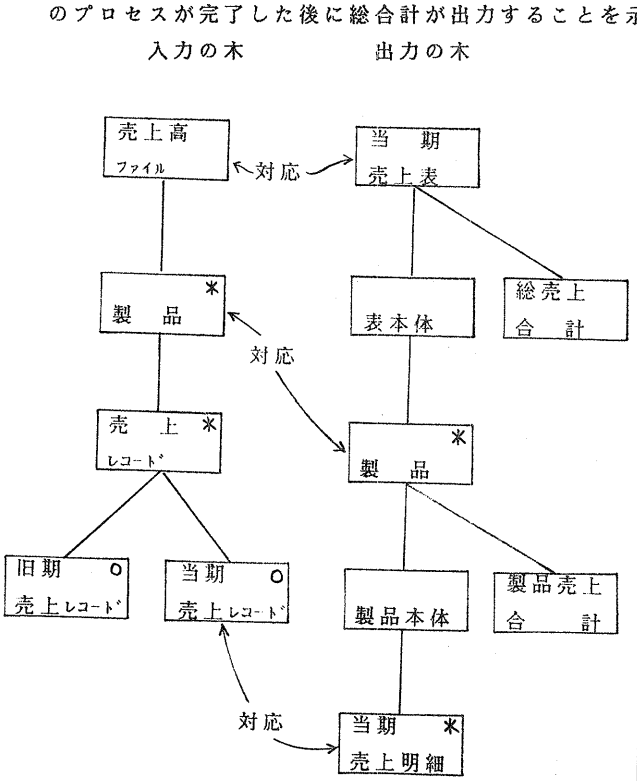


図-6　データ構造図

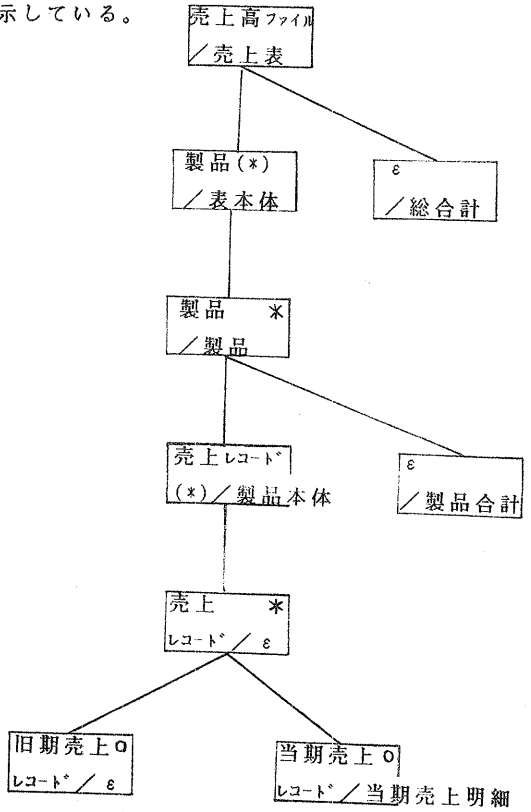


図-7　プログラム構造図

さらに，図-7のプログラムの木を順次入出力状態推移図に変換する。（図-8）

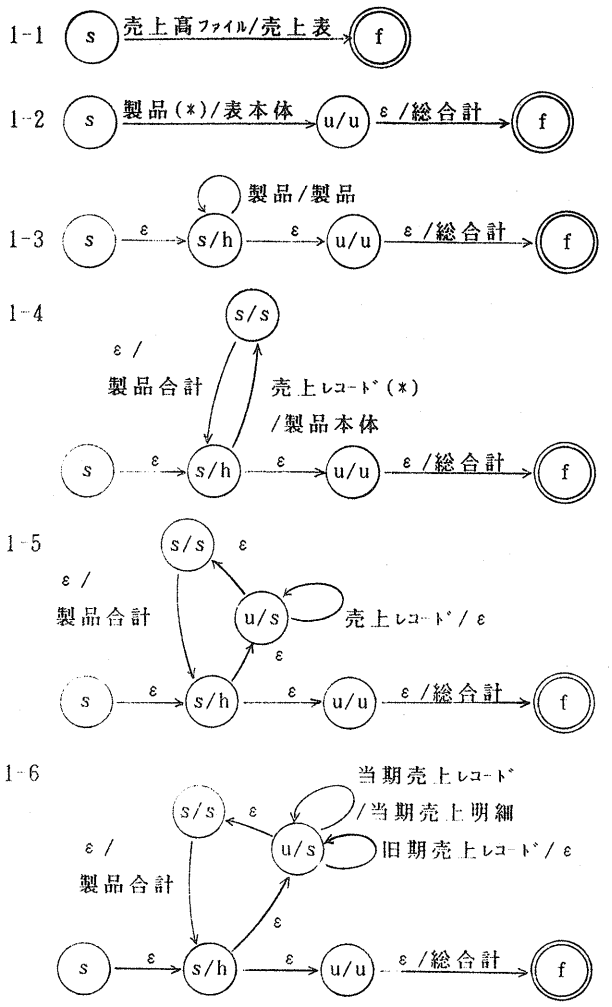


図-8　入出力状態推移図の作成

図-8を簡略化して結局図-9 が得られる。

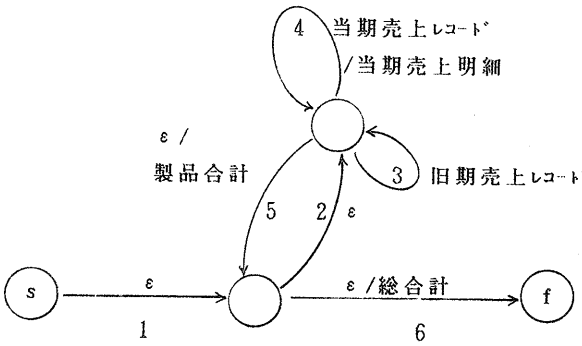


図-9 入出力状態推移図

当期売上ﾚｺｰﾄﾞ/当期売上明細は，当期売上ﾚｺｰﾄﾞが入力したときに当期売上明細が出力することを示している。ε，旧期売上ﾚｺｰﾄﾞは，各々ε/ε,旧期売上ﾚｺｰﾄﾞ/εの省略形である。

入出力状態推移図の開始ノードｓから終了ノードｆまで辿ることによって入力データと対応する出力データの組合わせを組織的に準備することができる。

この例題でテスト・ケースを選択してみよう。図-9 において各アークに番号が付いているが，この数値を列挙することによって，入出力データの組合わせの軌跡を辿ることができる。

すべてのアークを網羅すべきであるというパス・カバレッジの観点からすれば図-9 から，つぎのテスト・ケースが得られる。

ケース1　ｓ　1　2　3　4　5　6　f

われわれは経験上，それ以外のパスを選択することがテスト・データとして有効であることを知っている。

繰返し要素，即ち，同じノードから出発し終了する一連のアークをもつノードに関して0回ループを選択することは有意義である。即ち，

ケース2　ｓ　1　6　f　[1]

さらに，繰返しアークを2回以上，必要があれば最大回数ループすることも同様に有効である

選択要素，即ち，2本以上の外向きのアークをもつノードに関しは，1本のアークのみを選択するケースを選択したい。図-9において，

ケース3　ｓ　1　2　3　5　6　f
ケース4　ｓ　1　2　4　5　6　f

結局，テスト・ケース1，2，3，4が得られた。パス・カバレッジの観点からすれば，テスト・ケース1が基本である。その他に経験上からテスト・ケース2，3，4を追加した。開始ノードｓから終了ノードｆに至る各々のテスト・ケースは，独立のファイルである。多数の独立のテスト用ファイルを準備するのは煩雑であるから，ケース1，3，4をひとつにまとめてしまうと新しいテスト・ケース5が得られる。

ケース5　ｓ　1　2　3　4　5　2　3　5　2　4　5　6　f

---

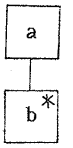
[1]　図-9のパスからは，

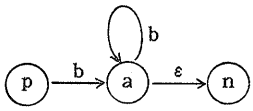
ｓ　1　2　5　6　f

が得られるが，プログラムの意味を考えると，このテスト・ケースはあり得ないことが分かる。

状態推移図のうえで通らないパスを識別することは煩雑であるから，これを避けるためにはつぎのように考えてもよい。

通らないパスが発生するのはＪＳＰでは繰返し要素の構成要素の発生件数を0件以上の場合と1件以上の場合を通常区別していないからである。構成要素の件数が1件以上と保証されている場合には以下のように書く。





この表記をこの例題に適用する（”製品”に属する”売上レコード”は1件以上かならず存在する）と，通らないパスがなくなることが容易に分かる。

5．結論

テスト・ケース選択法には2種類ある。ひとつはプログラムのコードに基づくもの（ホワイトボックス法）であり，他方はプログラムの仕様に基づくもの（ブラックボックス法）である。プログラム・テストに関する論文の多くのものは前者によるものであって，後者によるものは限られている。その理由は，ほとんどのプログラム仕様が形式的に書かれていないために，ブラックボックス法でテスト・データをつくることが大変困難であるからである。

ここで提案した方法を再度要約すると以下のようになる。ＪＳＰ流のデータ構造が課題を反映したものであるという事実に着目して，プログラムの課題を入力領域を出力領域に変換する変換系と解釈する。更に，入力領域，出力領域を正規言語とみなし，正規文法を満たす文章としてテスト・ケースを選択する。このとき，正規文法から変換された状態推移図のアークをすべて網羅するパス・カバレッジを基本に，更に，若干のプログラムの意味を考えて，テスト・ケースを選択する。選択されたテスト・ケースはプログラムが変換系を実現しているかどうか，即ち，入力の終端記号（入力の木の葉）を出力の終端記号（出力の木の葉）に変換するプログラムの木の葉がプログラム中に適切に実現されているかどうかをテストするのに役立つ。つまり，作成されたテスト・データがプログラム内部のパスを網羅していれば，そのプログラムは良いプログラムであると見なすことができる。したがって，テスト実行時にプログラムの木の葉に関するパス・カバレッジ測定ツールを用いれば一層効果的となろう。

最後に，適切な助言をいただいた山崎利治，前田英次郎，染谷　誠各氏に感謝する。

＊参考文献

［1］M.A. ジャクソン著，鳥居宏次訳：構造的プログラム設計の原理，日本コンピュータ協会，東京（1980）
(M.A.Jackson : Principles of Program Design, ACADEMIC PRESS, P.299 (1975))

［2］峰尾欽二，プログラミング方法論（ジャクソン法），情報処理，Vol.23，No.11，pp.1063～1074