Vol. 27 No. 7 情報処理学会論文誌 July 1986

ソフトウェア設計自動化のための仕様記述言語 F†

鎌坂恒夫†† 阿草清滋†† 大野 豊††

ソフトウェア設計の生産性向上のため計算機支援による自動化が望まれる．そのためには設計工程を形式化かつ単純化する必要がある．本研究では設計工程への入力を問題の入出力データ定義，工程からの出力をその入出力データ間の写像を実現する関数記述とする．このような工程における仕様記述には関数型言語が向いているが，従来の関数型言語ではデータに関する仕様が記述できないので，それを補って設計に適した仕様記述言語Fを定義する．F言語ではデータおよび関数はともに結合子とよぶ作用素を用いて形式的に記述される．それによってデータと関数との対応規則を定式化することができる．この対応規則を与えられた入出力データの仕様に繰り返し適用することによって，目的とするソフトウェアの処理構造を機械的に設計することができる．データ結合子はデータ構造を表現し，関数結合子は構造変換を行うものであるから，この対応規則はジャクソン法の設計技法を定式化した規則といえる．また関数合成を含む規則は，構造変換過程における中間データ構造を設定することにより，複合／構造化設計の STS 分割技法を形式化したものである．導かれた関数記述は，未定義部分をスタブとして抽象実行を行うことができる．すなわち，F言語による設計仕様は目的ソフトウェアの初期プロトタイプとなる．

1. はじめに

ソフトウェア設計のコストを軽減しソフトウェアの品質を一定の水準に保つためには，計算機による自動化が可能となるように設計工程を明確に規定する必要がある．ソフトウェア設計の方法論には有名なものがいくつかあり実際の設計作業にもある程度応用されているが，それら方法論は設計作業の規範を述べたものであって設計自動化に結びつけるには形式化が不十分である．例えば複合／構造化設計[7]では，その基本技法である STS 分割における「最大抽象点」の定義があいまいでその設定は設計者の判断に依存する．抽象データ型を用いる設計技法においては，与えられた問題の中から抽象化すべきデータを選定する指針が不明確である．また，ジャクソン法[5),6)]ではデータ構造にプログラム構造を対応させる規則が十分形式化されていない[4]．

設計工程への入力は問題仕様であるが，問題仕様の記述に用いられる概念要素は問題ごとに異なるものが多く，そのすべてを列挙して定義することは不可能である．設計工程を自動化の可能なものとするためには，汎用性を多少犠牲にしても工程への入力を限定しなければならない．ジャクソン法はこの点では示唆を与えている．

本研究では設計工程への入力は設計すべきソフトウェアの入出力データの定義とする．データ定義に限定する理由は，形式的な記述が可能であり，とくに本論文において主な対象問題領域としている事務処理ソフトウェアに対しては，問題の仕様化に必要となる要素のうち最も早い時期にその具体的なイメージが明確になると考えられるからである．工程からの出力は，与えられた入出力データ間の写像を実現する関数の仕様である．この仕様は抽象実行可能であり，設計すべきソフトウェアの初期プロトタイプとなる．データおよび関数をともに結合子 (combinator) とよぶ作用素を用いて形式的に記述し，それらの間の対応規則を定式化することによって，ソフトウェア設計自動化の基礎を与えるのが本研究の趣旨である．そのための設計仕様記述言語がここで提案するF言語である．

2. 設計仕様記述言語 F

2.1 基本概念

ソフトウェア設計のための仕様記述言語Fは以下のような概念要素を基礎にしている[1]．

オブジェクト

オブジェクトはアトム，またはオブジェクトを要素とする列である．アトムの集合は設計の詳細化のレベルに応じて任意である．オブジェクトの全体集合は他の概念要素を定義する基礎となる集合であるが，F言語ではオブジェクトを直接表現することはない．

データ型

データ型は個々の問題に固有の特徴によってクラス分けされるオブジェクトの集合である．あるい

† Specification Description Language F for Software Design Automation by TSUNEO AJISAKA, KIYOSHI AGUSA and YUTAKA OHNO (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室

はその集合を規定する述語と考えてもよい.

データ結合子

データ結合子はデータ型をデータ型に写像する作用素である. **表1**に示すように連接, 反復, 選択の三つのデータ構造を構成する.

関数

関数はオブジェクトをオブジェクトに写像する単項演算子であって, その定義域, 値域はデータ型によって定義される. 命題関数 (述語) もこの関数に含まれる.

関数結合子

関数結合子は関数を関数に写像する作用素である. これは Backus の FP[2] における関数形式 (functional form)[3] と同様の概念である. 連接, 選択, 合成などの基本的なものを**表2**にあげる.

2.2 関数型仕様記述言語とソフトウェア設計原理

ソフトウェアは入力データから出力データへの写像を実現するものとみなすことができる. 関数型言語はそのような見方を反映しており, 汎用性の高い機能要素を小さな言語の枠組の中にまとめている[2]ため, 設計自動化の見通しがたてやすいと考えられる.

無変数関数型言語による結合子を用いた関数記述は, 複合/構造化設計やジャクソン法に示された分割技法によるモジュール構造を, 以下のように形式的, 系統的に表現することができる. (ただし, f,g,h は関数, p,q は命題関数 (述語) である.)

- STS (Source/Transform/Sink) 分割

  $f.g.h$ (合成結合)

- トランザクション分割

  $\{p \to f;$
  $q \to g;$
  ……
  $h \quad\}$ (選択結合)

- データ構造分割 (ジャクソン法)

  $[f,g,\cdots,h]$ (連接結合)
  $R(f,g)$ (縮約結合)
  $A(f)$ (分配結合)
  $\{p \to f; q \to g; \cdots; h\}$ (選択結合)

すなわち, 複合/構造化設計およびジャクソン法の技法を統合して用いることができ, 前者をモジュール構造設計, 後者をプログラム論理設計と限定する必要はない. 結合されている各関数を繰り返し分割することにより, 概略設計から詳細設計まで一貫して進めることができる.

表1 データ結合子
Table 1 Data combinators.

データ結合子	記法	結合されたデータ型が含むオブジェクト
連接結合子	["x", …, "z"]	各オペランド型のオブジェクトを要素とする列オブジェクト
反復結合子	["x"…]	オペランド型の任意個のオブジェクトを要素とする列オブジェクト
選択結合子	{"x", …, "z"}	各オペランド型の和集合に含まれるオブジェクト

表2 基本的な関数結合子
Table 2 Primitive function combinators.

関数結合子	記法	結合された関数をオブジェクト X に作用させたときに返されるオブジェクト
連接結合子	$[f,\cdots,g]$	$\langle f: X,\cdots,g: X\rangle$
縮約結合子	$R(f,g)$	$X=\langle X_1,\cdots,X_n\rangle$ のとき $f: \langle X_1,\cdots,f: \langle X_n,g: \langle\ \rangle\rangle\cdots\rangle$
分配結合子	$A(f)$	$X=\langle X_1,\cdots,X_n\rangle$ のとき $\langle f: X_1,\cdots,f: X_n\rangle$
選択結合子	$\{p \to f; g\}$	$p: X$ が真のとき $f: X$ そうでないとき $g: X$
合成結合子	$f.g$	$f: (g: X)$

注 1) $\langle X_1,\cdots,X_n\rangle$ は列オブジェクトを表す.
注 2) $f: X$ は関数 f をオブジェクト X に作用させた結果のオブジェクトを表す.
注 3) 縮約結合, 分配結合された関数をアトムに作用させたとき, および p が命題関数でないとき, 結果は定義されない.

しかし, FP のようにデータ型の概念を排除してすべての関数が同一の定義域と値域をもつ体系では, 設計段階において重要な関数間インタフェースの記述ができず, 定義域, 値域の詳細化によって設計を詳細化していくことも困難である. そこでF言語では関数の定義域, 値域を与えるデータ型の概念を導入し, 関数を結合子を用いて構成するのに対応してデータ型も結合子を用いて構成する体系とする. それによって3章で述べるようにデータと関数との対応規則が定式化され, 自動化の可能なデータ指向の設計工程が確立される.

次の2節で述べるように, F言語はデータ仕様記述言語と関数仕様記述言語からなる (付録参照). 前者は設計工程への入力言語であり, 後者は出力言語である. データ仕様は設計者によって記述されるが, 関数仕様は計算機支援システムによって出力されることを

目標としている.

2.3 データ仕様

設計工程への入力となる目的ソフトウェアの入出力データの定義は，F言語によってデータ仕様として記述する．F言語によるデータ仕様は次のように定義される.

- データ仕様はデータ定義式の集合である.
- 〈データ定義式〉::=〈データ型〉==〈データ結合式〉
- データ型はその名前を二重引用符で括って表記する.
- $\delta, \delta_1, \cdots, \delta_n$ をデータ型とするとき,

  $\delta, [\delta_1, \cdots, \delta_n], [\delta \cdots], \{\delta_1, \cdots, \delta_n\}$

はデータ結合式である.

データ仕様は "domain", "range" を左辺に持つデータ定義式を必ず一つずつ含まなければならない．それら以外のデータ定義式の左辺のデータ型は，データ仕様に含まれる少なくとも一つのデータ定義式の右辺に現れなければならない．すなわち，データ定義式はデータ型の生成規則であって，データ仕様によって記述されるものは，"domain", "range" の二つのデータ型，すなわち目的ソフトウェアの入出力データの型である.

データ定義式の左辺に現れないデータ型をとくにソートとよぶ．ソートはアトムの集合であり，アトムの集合が任意であるのと同様，ソートの集合も設計の詳細化のレベルに応じて任意である．なお，データ定義式の右辺が単独のデータ型であるとき，左右両辺のデータ型は同一とみなす.

データ型の構成作用素が連接，反復，選択の三つのデータ結合子であることから，正則集合を連想させるが，オブジェクトの集合であるデータ型は正則集合ではない．データ型は連接，反復（閉包），選択（和集合）について閉じているが，連接データ結合子は正則集合における連接作用素と異なり，結合則に従わない．例えば

"d1"==["a", "b", "c"]

と

"d2"==["a", "x"]

"x"==["b", "c"]

（データ結合式の入れ子の形に書けば，["a", ["b", "c"]]）

とは異なるデータ型を定義し，また，"d1" は

[["a", "b"], "c"]（これを "d3" とする）

とも異なる.

これは，一般に語の連接，閉包がまた語であるのに対して，アトムの連接，閉包（反復）は（列）オブジェクトであってアトムではないこと，オブジェクトの集合においては，その表現言語の要素としてアトムとともに列構成子が必要であることに対応している．このことは，データ構造，さらに一般に「構造」の表現に本質的な点であると考えられる．すなわち，列構成子がオブジェクトの並びに区切りを入れて「構造」を与えているのである．データ型に対する連接データ結合子は，表1に示したように，オブジェクトに対する列構成子に対応する.

"d1", "d2", "d3" は同じ正則のデータ言語を表現するという形式化[4]もありうるが，それではソフトウェアの入出力データはすべてバイト列であるとみなすような低レベルの定式化になる．ソフトウェアの設計段階においては，より抽象化レベルの高い論理的なデータ構造を記述できなければならない.

図1にF言語によるデータ仕様の記述例を示す．例題は次のとおりである．あるトランザクション・ファイルは，注文のあった製品番号 (prodid), 単価 (price), 数量 (qty), 決済未決済の別 (stat) を顧客 (cust) ごとに記録している．これを入力し，各顧客についての発注件数 (nrec) と未決済残高 (debt), およびそれらの総計 (totrec, totdebt) を出力する.

2.4 関数仕様

設計工程から出力される結果は，与えられた入出力データ間の写像を実現する関数の仕様である．これは以下のように定義されるF言語による関数仕様の形に表される.

- 関数仕様は関数定義式の集合である.
- 〈関数定義式〉::=〈関数〉:〈定義域〉 =〉〈値域〉 ==〈関数結合式〉
- 関数はその名前によって表記する.
- 定義域，値域はデータ型またはデータ結合式である.

```
"domain" == ["cust"...]
"cust" == ["custid","body"]
  "body" == ["trec"...]
  "trec" == ["prodid","price","qty","stat"]
    "stat" == ("paid","notyet")
"range" == ["sumbody","totrec","totdebt"]
  "sumbody" == ["custsum"...]
    "custsum" == ["nrec","debt"]
```

図1 データ仕様記述例1
Fig. 1 Example of data specification 1.

●関数結合式は次のように定義される.

i) 関数は関数結合式である.

ii) $\phi_1, \phi_2, \cdots, \phi_n$ を関数結合式, ψ を命題関数を結果とする関数結合式, Φ を連接, 選択, 合成以外の関数結合子とするとき,

$[\phi_1, \cdots, \phi_n]$ $\{\psi \to \phi_1 ; \phi_2\}$,

$\phi_1 . \phi_2$ および $\Phi(\phi_1, \cdots, \phi_n)$

は関数結合式である.

関数仕様は main を左辺にもち, その定義域, 値域が "domain", "range" である関数定義式を必ず一つ含まなければならない. それら以外の関数定義式の左辺の関数は, 関数仕様に含まれる少なくとも一つの関数定義式の右辺に現れなければならない. すなわち, 関数定義式は関数の生成規則であって, 関数仕様によって記述されるものは, "domain", "range" を定義域, 値域とする関数 main, すなわち与えられた入出力データ間の写像を実現する目的ソフトウェアの仕様である.

関数定義式の左辺に現れない関数はいわゆる原始関数であるが, アトムやソートと同様に原始関数の集合も設計の詳細化のレベルに応じて任意である.

関数結合式の性質をその定義域, 値域との関連において公理化すると, 次のようになる.

(F1) f: "x"⇒"y"==[$f_1, \cdots, f_n$] ならば,
"y"==["y_1", ⋯, "y_n"]
ただし,
f_i: "x"⇒"y_i" ($i=1, \cdots, n$)

(F2) f: "x"⇒"y"=={$p \to f_1 ; f_0$} ならば,
"x"=={"p_1", "p_0"}
"y"=={"y_1", "y_0"}
ただし,
p: "x"⇒"bool" ("bool" は真理値集合)
"p_1" は p を特性関数とする集合
"p_0" は "x" に関する "p_1" の補集合
f_i: "p_i"⇒"y_i" ($i=0, 1$)

(F3) f: "x"⇒"y"==$f_1 . f_2$ ならば,
"z" があって,
f_2: "x"⇒"z"
f_1: "z"⇒"y"

(F4) f: "x"⇒"y"==$A(f_1)$ ならば,
"x"==["x_1"⋯]
"y"==["y_1"⋯]
ただし,
f_1: "x_1"⇒"y_1"

(F5) f: "x"⇒"y"==$R(f_1, f_2)$ ならば,
"x"==["x_1"⋯]
ただし,
f_1: ["x_1", "y"]⇒"y"
f_2: "nil"⇒"y"
("nil" は空列だけを要素とするデータ型)

```
main: "domain" => "range"
  == [p_sumbody,p_totrec,p_totdebt]
p_sumbody: "domain" => "sumbody"
  == A(p_custsum)
p_custsum: "cust" => "custsum"
  == [p_nrec.body,p_debt.body]
p_nrec: "body" => "nrec"
  == R(f_nrec,g_nrec)
p_debt: "body" => "debt"
  == R(f_debt,g_debt)
f_debt: ["trec","debt"] => "debt"
  == < is_paid.stat.trec -> f_debt_paid; f_debt_notyet >
p_totrec: "domain" => "totrec"
  == R(f_totrec,g_totrec)
p_totdebt: "domain" => "totdebt"
  == R(f_totdebt,g_totdebt)
```

図 2 関数仕様記述例 1
Fig. 2 Example of function specification 1.

図1のデータ仕様に対応する関数仕様を**図2**に示す. この関数仕様の構成については3.1節で詳述する. なお, body や stat のように, その定義域を与えるデータ結合式に現れるデータ型と同名の関数は, 連接構造の要素をとりだす原始関数で, 名前関数とよんでいる. FP のセレクタ関数 **2,4** にそれぞれ対応するが, それらに比べて視認性, 了解性にすぐれている. これは, セレクタ関数が, λ-計算に基づく体系における束縛変数の結合操作を一般の関数の作用として統合したものであるからである.

3. F 言語に基づくソフトウェア設計

3.1 データと関数の基本対応規則

2.4 節に述べた関数結合式の性質は, データと関数との対応規則を定式化したものとみなすことができる. F言語に基づくソフトウェア設計の基本的な方法は, この対応規則に従って, 設計工程に入力される入出力データ仕様からその間の写像を実現する関数仕様

を機械的に導くものである[1]．設計はデータ仕様の構成に従ってトップダウンに進められる．

（1） “domain” および “range” を定義域，値域とする関数 main を設計する．すなわち関数定義式の右辺になる関数結合式を対応規則に従って導く．結合式のオペランドとなる関数の定義域，値域も対応規則に含まれており，この時点で決まる．

（2） すでに導かれた関数結合式に現れる未設計の関数を，その定義域，値域を与えるデータ結合式に基づいて設計する．ただしその関数に対して関数結合式が導けなければ（定義域，値域ともソートである場合など），その関数は未定義であるとする．

（3） 未設計の関数がなくなるまで(2)を繰り返す．

2.4 節の公理(F1)～(F5)は，関数結合式を前提としてその定義域，値域をきめるデータ結合式を与えており，直観とも一致しているが，上述の設計法においてはこれを逆に用いている．したがって，この設計法が適用できるのは，(F1)～(F5) の逆 ((F1′)～(F5′)) もまた公理とし，それらをデータと関数との基本対応規則とする問題領域である．

基本対応規則のうち必ずしも直観と一致しないようにみえるのは（F4′）である．例えばソーティングの入出力データはともに反復構造であるととらえることができるが，ソーティングを分配結合で実現することはできない．このような反復構造の要素の相対位置を変えるような問題の場合，ここではデータ仕様の定義が不適切であるとする立場をとる．ソーティングの例では，その入出力データは，例えば，

“domain”=={“nil”, “seq”}
“seq”==[“e”…]
“range”=={“nil”, “sorted”}
“sorted”==[“minimax”, “range”]

のように与えられるべきであるとする．

基本対応規則をデータ側からみた ((F1′)～(F5′)) 場合，定義域，値域を与えるデータ結合式のあらゆる組み合わせが含まれているわけではないが，例えば定義域が反復構造，値域が連接構造である場合には，(F1′) を適用した後（F4′）または（F5′）を使って設計を進めればよい．（F2′）においては “y_1” と “y_0” が同じデータ型である場合（多くは同じソートである場合），すなわち値域が選択結合とならない場合もある．しかし値域が選択結合であれば，その選択を決定する情報は入力データに含まれているべきであり，定義域も選択結合でなければならない．これは，選択に用いられる条件はプログラムの意図によって命題関数として決定されるのではなく，本来述語であるデータ型の定義によって問題の意味として与えられるべきものであるという，データ（述語）と関数を分離する立場である．値域が反復構造で定義域がそうでない応用例（反復構造の生成）は比較的出現頻度が低く，反復を生成する情報の与え方が一定でないため規則化していない．定義域が連接構造である場合の問題は後で触れる．

例として図1のデータ仕様にこの基本対応規則を適用し，図2の関数仕様を導いてみよう．まず main は “range” が連接構造であるから（F1′）に従って設計される．次に，p-sumbody は定義域，値域ともに反復構造であるから，（F4′）によって分配の関数構造が採用される．p-custsum は main と同様（F1′）に従う．次に設計が進められる p-nrec と p-debt には，反復構造を単純型に変換する縮退の関数構造が（F5′）に従って与えられる．p-totrec, p-totdebt も同様に（F5′）による．f-debt の選択結合は，連接と選択のデータ結合子に関する分配法則

[“a”, {“b”, “c”}] = {[“a”, “b”], [“a”, “c”]}

を f-debt の定義域に2回適用した後，（F2′）によって導かれる．

このように，基本対応規則に従って入出力データ仕様から関数仕様をほぼ機械的に導くことができる．ただし，上の例で，p-nrec と p-debt の定義域は，p-custsum の設計の段階では “body” ではなく “cust” である．この問題は定義域が連接構造である場合に常に起こるが，ここでは設計者の介入が必要で，p-nrec や p-debt の行う処理に “custid” は不要であるとの判断のもとに，それらにセレクタ関数（名前関数）を合成するように設計しなければならない．

ここで用いた基本対応規則は定義域，値域それぞれ一つずつのデータ結合式に注目し，それらを構成するデータ結合子の組み合わせによって対応する関数結合子を決定する．データ結合子はジャクソン流のデータ構造を表現しており，またこの対応規則に現れる関数結合子は基本的な構造変換を行うものである．したがって本節で述べた基本対応規則はジャクソン法の基本技法を定式化した規則であると考えられる．

3.2 基本対応規則の拡張

前節で述べたように，データ結合式の多重構成であるデータ仕様に対応する関数仕様は，多くの場合，各データ結合式ごとに基本対応規則に従って得られる関

数結合式を，トップダウンに多重構成にしたものとなる．しかし，注目している定義域，値域のデータ結合式について，その要素となるデータ結合式を考慮にいれる必要がある場合もある．

表3は，定義域，値域を与える2レベルまでのデータ結合式の構成と，関数結合式の構成との対応表である．表3の見出しの c, i, s はそれぞれ連接，反復，選択のデータ結合子によって構成されるデータ結合式を一般的に表し，ci などは結合式の2レベル構成を表す．例えば，ci は

[["*a*"…], "*b*"],
["*a*", ["*b*"…], ["*c*"…]]

など連接データ結合式のオペランドとして反復データ結合式を含むデータ結合式を一般的に表す．

表3の項目中の C, S, R, A はそれぞれ連接，選択，縮約，分配の関数結合子によって構成される関数結合式を一般的に表し，CRS などは関数結合式の多重構成を表す．例えば，CA は

$[A(f), g]$, $[f, A(g), A(h)]$

など連接関数結合式のオペランドとして分配関数結合式を含む関数結合式を一般的に表す．Dについては後述する．

表3はそのいくつかの見出しが省略され簡約化されている．これは（F2′）について前節で述べたように，値域が選択結合ならば定義域も選択結合であり，値域が選択結合である場合の関数の形は値域側の選択結合を除いたものに対する関数の形と同じであること，および，選択結合子の結合則，選択結合子と連接結合子の分配則に基づいている．一は反復構造の生成にあたる変換であり，ここでは規則化しない．

表3から明らかなように，データ結合式の2レベル構成の定義域，値域に対応する関数結合式の構成は，多くの場合，各データ結合式ごとに対応する関数結合式を前節の基本対応規則に従って単純に組み合わせたものになる．このことは3レベル以上の構成についても同様である．

2レベル構成の定義域，値域間の変換を一挙に行わなければならないものとして，ここで新たに必要となったのはDである．これは ci から ic への変換，例えば，

表3 定義域，値域と関数との対応表
Table 3 Correspondence table between domain/range and function.

定義域＼値域	ソート	c	i	cc	ci	ic	ii
c	f(*)	C	—	CC	—	—	—
i	R	CR	A	CCR	CA	AC	—
s	S	CS	—	CCS	—	—	—
cc	f(*)	C	—	CC	—	—	—
ci	R. D	CR. D	A. D	CCR. D	CA. D	D	—
cs	S	CS	—	CCS	—	—	—
ic	R	CR	A	CCR	CA	AC	—
ii	R. AR	CR. AR	AR	CCR. AR	CAR	ACR	AA
is	RS	CRS	AS	CCRS	CAS	ACS	—
si	SR	CSR	SA	CCSR	CSA	SAC	—

(*) 結合されない関数

```
"domain" == ["docfile","request"]
  "docfile" == ["document"...]
    "document" == ["title","docdata","keyws"]
      "keyws" == ["keyword"...]
  "request" == "keyword"

"range" == ["title"...]
```

図3 データ仕様記述例2
Fig. 3 Example of data specification 2.

[["*a*"…], "*b*"]⇒[["*a*", "*b*"]…]
["*a*", ["*b*"…], ["*c*"…]]⇒[["*a*", "*b*", "*c*"]…]

などの変換を行う関数からなる関数結合式を一般的に表す．

表3の項目でもう一点基本対応規則（公理）の真の拡張となっているのは，合成結合子を用いた規則である．これらの規則は経験則に基づく（F3′）の詳細化である．合成結合子はその右オペランド関数が出力したオブジェクトに左オペランド関数を作用させるものであるから，これはデータの流れあるいはデータの変換過程の分析に基づく設計という意味で，2.2 節に述べたように STS 分割を指示するものである．

図3にデータ仕様を示す文献データ検索の例題は，入力データが ci 型，出力データが i 型である．表3によれば，ic 型の中間データ

[["document", "request"]…]

を設定することによって（**図4**参照），

A. D

という関数構造が与えられ，定式的にモジュール分割

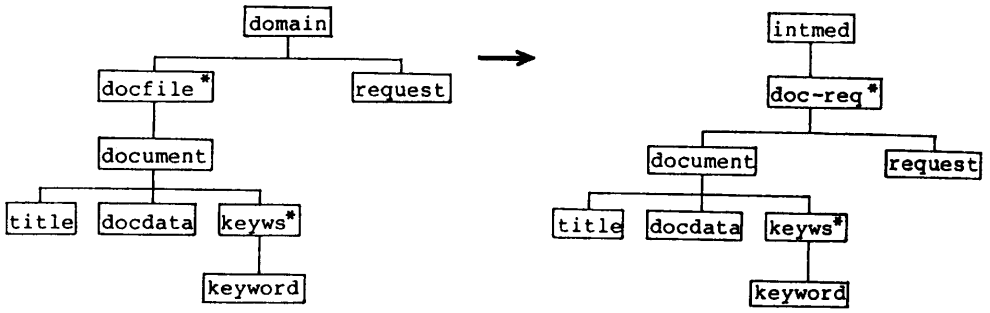


図 4 中間データへの構造変換例
Fig. 4 Example of transformation into intermediate data.

が行われる.

この例でみた ci 型は反復構造が一つのものであったが, ci 型のデータには複数の反復構造からなるものもあり, 多くの型が含まれる. したがって, D による変換は, 例えば,

["a", ["b"…], ["c"…]]

に対して,

[["a", "b", "c"]…]

[["a", "b", ["c"…]]…]

など, 一意でない. ci 型の定義域を持つ与えられたデータ仕様に対して, Dによる適切な中間データを設定するためには, ci 型の反復構造の要素について調べる必要があり,

- ci 型の複数の反復構造の要素がそれぞれどのような下位構造を持つか.
- ci 型の反復構造の要素が値域側においてどのような構造のどのレベルに現れるか.

等をもとに, ある程度の規則化が可能である.

ジャクソン法でとりあげられている構造不一致のうち, 境界不一致, 脈絡不一致の問題も合成結合を用いた STS 分割の考え方による設計となる. (順序不一致の問題は, 前節で述べたソーティングにおける問題と同様の問題を含む.) 脈絡不一致は TSS システムのシステムログの集計問題にみられるように, is 型から i 型への変換であるが, 表3に従って AS を用いた変換だけでは正しい解を与えない. 同じユーザに対するログオンデータとログオフデータが対応していることを is 型のデータ結合式に表現できないからで, ログオンデータとログオフデータを振り分けた ci 型のデータ, それらをユーザごとに対応させた ic 型のデータなどを中間データとして与えることにより, その問題の意味が明らかになる.

境界不一致の問題も中間データを必要とし, 関数合成によって設計される. ただし, 例えば電報解析問題[5]において, ブロックを単なる語の反復構造 (i 型) であるとせず, 電報の終わりを示す語と一般の語を区別する (is 型) ことによって, 不一致の存在を示唆することができる. 境界不一致の存在が判明すれば, 中間データの設定も含めてその処理はかなり定型化できる.

以上のように表3は, データ構造の分析に基づくジャクソン法とともに, データの流れの分析に基づく STS 分割技法による設計過程をも定式化する足掛かりを与えたものであるといえる. 後者については, 与えられた入出力データ構造から, その変換過程において必要となる中間データ構造をある程度自動的に決めることにより, 明確なモジュール境界を与えている.

4. F 言語による設計仕様の抽象実行

本論文で定義したデータ仕様には, ソートやソート間関係の仕様が含まれていない. したがって, 構造をもたないアトミックなデータ間 (表3に(*)で示したように定義域がソートの連接構造である場合も含む) の値の変換処理を行う関数を自動的に設計することはできない. 図2の f-debt-paid や f-debt-notyet はその例で, "debt" は "price" と "qty" の積の総和であるといったデータの意味の定義を必要とする. しかし, そのような未定義の部分は局所化されているので, 記号的なオブジェクトを返すスタブ[8]として処理することにより, 関数仕様の抽象実行が可能である.

関数仕様を構成する関数結合子は関数に対する作用素であるから, 実行にあたってはこれをオブジェクトに対する演算に変換する必要がある. また, 名前関数はデータ仕様を参照してセレクタ関数に変換しなければならない. 実行はそれらの演算や関数を関数 main から始めて順次, 入力オブジェクトに適用していくこ

```
< < <nrec-by-f_nrec  debt-by-f_debt>
    <nrec-by-f_nrec  debt-by-f_debt>
    <nrec-by-f_nrec  debt-by-f_debt>
  >
  totrec-by-f_totrec
  totdebt-by-f_totdebt
>
```

図5 抽象実行結果の例
Fig. 5 Result of example abstract execution.

とによって進められる．連接結合の実行にはオブジェクトスタックとレベル管理スタックを用いている．入力オブジェクトについては，入力データ仕様を満たす簡単なオブジェクト例が実行系によって自動的に与えられる．定義されていない関数はオブジェクトに直接作用し，記号オブジェクトを結果として与える．

図2の関数仕様を抽象実行した結果を図5に示す．(反復構造データは要素数3の列オブジェクトとして与えられている．) 記号オブジェクトは，そのデータ型名とそれを結果とする関数名によって表記される．機械的に設計されたこの段階の仕様の抽象実行結果からは，入出力データ間の構造変換が正しく行われていることが確認できる．未定義の関数に算術演算や関係演算の原始関数を用いた簡単な関数を割り当て，この仕様を詳細化すると，具体的な入力オブジェクトに対して値の変換処理まで含めた具体的な実行結果を与えるようになる．すなわちF言語による設計仕様は，目的ソフトウェアに対する，進化の容易な初期プロトタイプとなっている．

5. おわりに

ソフトウェア設計の自動化のためには，設計されるソフトウェアの標準化とともに設計工程の標準化が必要である．この観点から，本研究では工程への入力を目的ソフトウェアの入出力データの仕様，工程からの出力をその間の写像を実現する処理の関数的仕様とし，それらの間の対応規則を定式化することのできる設計仕様記述言語Fを定義した．また，その定式化された対応規則に基づく機械的なソフトウェア設計法について述べた．筆者らは3章に述べた表3に基づく自動化設計システム，および4章で述べた抽象実行システムをすでに開発し，現在評価中である．(システムの記述言語はC，ホストは ECLIPSE MV/6000 AOS/VS である．)

F言語ではデータおよび関数をともに結合子という構成作用素 (forming operator)[3] を用いて形式的に記述することによって，それらの対応規則の定式化を可能なものにしている．さらに3.2節に述べた表3は，対応規則の拡張についても規則性のあることを示唆している．すなわち，より高度な設計自動化のためには，多数のデータ定義式からなるデータ仕様を，ソートに関する仕様なども含めてパターン化し，それに対応する関数仕様のパターンをきめる規則を定式化する必要がある．しかしそのような関数仕様を少数の構成的な規則に基づいて導出するのは困難であり，仕様のデータベース化の方向に進むのが妥当であると考えられる[9]．本論文はその基礎を与えたものである．

謝辞 抽象実行システムおよび自動化設計システムの開発にあたっては，宮崎雅也氏 (松下電器)，岡村和男氏 (松下電器) の協力を得ており，ここに謝意を表する．

参考文献

1) Ajisaka, T., Agusa, K. and Ohno, Y.: Integral Software Development through a Functional Language, *Proc. of International Symposium on Current Issues of Requirements Engineering Environments*, Ohm/North-Holland, pp. 33-38 (1982).
2) Backus, J.: Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs, *Comm. ACM*, Vol. 21, No. 8, pp. 613-641 (1978).
3) Backus, J.: Function Level Programs as Mathematical Objects, *Proc. of the 1981 Conference on Functional Programming Languages and Computer Architecture, ACM*, pp. 1-10 (1981).
4) Hughes, J. W.: A Formalization and Explication of the Michael Jackson Method of Program Design, *Softw. Pract. Exper.*, Vol. 9, pp. 191-202 (1979).
5) Jackson, M. A.: *Principles of Program Design*, Academic Press, New York (1975).
6) M. Jackson Systems Ltd.: The Design of Programs and Systems, Tutorial Notes of 5th Int'l Conf. on Software Eng. (1981).
7) Myers, G. J.: *Composite/Structured Design*, Van Nostrand Reinhold, New York (1979).
8) 国友義久：効果的プログラム開発技法，近代科学社，東京，pp. 155-157 (1979).
9) 鯵坂恒夫，阿草清滋，大野 豊：関数スキーマベースを用いたソフトウェア設計自動化，情報処理学会論文誌，Vol. 26, No. 2, pp. 304-311 (1985).

付　　録

F言語の文法は以下に示すような LALR（1）文法である．

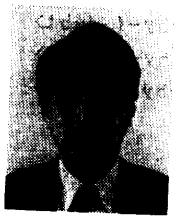
データ仕様記述言語

〈データ仕様〉→〈定義域仕様〉〈値域仕様〉
〈定義域仕様〉→"domain" == 〈データ結合式〉〈データ定義式並び〉
〈値域仕様〉→"range" == 〈データ結合式〉〈データ定義式並び〉
〈データ定義式並び〉→ε|〈データ定義式〉〈データ定義式並び〉
〈データ定義式〉→〈データ型〉==〈データ結合式〉
〈データ型〉→"〈名前〉"
〈データ結合式〉→〈データ型〉 |
　　[〈データ型並び〉]|
　　[〈データ型〉…] |
　　{〈データ型並び〉}|
〈データ型並び〉→〈データ型〉|〈データ型〉, 〈データ型並び〉

関数仕様記述言語

〈関数仕様〉→〈主関数定義式〉〈関数定義式並び〉
〈主関数定義式〉→main : "domain" => "range"==〈関数結合式〉
〈関数定義式並び〉→ε|〈関数定義式〉〈関数定義式並び〉
〈関数定義式〉→〈関数〉: 〈定義域〉 => 〈値域〉==〈関数結合式〉
〈関数〉→〈名前〉
〈定義域〉→〈データ結合式〉
〈値域〉→〈データ結合式〉
〈関数結合式〉→〈関数〉|
　　[〈関数並び〉]|
　　{〈関数結合式〉->〈関数結合式〉; 〈関数結合式〉}|
　　〈関数結合式〉. 〈関数結合式〉|
　　〈関数結合子〉 (〈関数並び〉)

（昭和59年6月18日受付）
（昭和61年4月17日採録）



鰺坂　恒夫（正会員）

1956年生．1980年京都大学理学部物理学系卒業．1982年同大学院工学研究科情報工学専攻修士課程修了．1985年同博士課程研究指導認定退学．同年より京都大学工学部情報工学科助手．ソフトウェア工学に関する研究に従事．とくにソフトウェア設計自動化，プログラム自動生成に興味を持つ．日本ソフトウェア科学会会員．

阿草　清滋（正会員）

1947年生．1970年京都大学工学部電気第二学科卒業．1972年同大学院工学研究科電気工学第二専攻修士課程修了．1974年より京都大学工学部情報工学科に勤務．現在，助教授．工学博士．ソフトウェア工学に関する研究に従事．とくに要求分析，ソフトウェア仕様化技法，ソフトウェア部品化技法，ソフトウェア開発モデルなどに興味を持つ．また，マンマシンシステム，分散処理システム等に関する研究も行っている．

大野　　豊（正会員）

1924年生．1946年東京大学工学部機械工学科卒業．同年より国鉄鉄道技術研究所に勤務，座席予約システム，新幹線運転管理システムなどの研究・開発に従事．1972年より京都大学工学部情報工学科教授，情報システム工学講座担任．工学博士．京都大学情報処理教育センター長を兼任．現在，ソフトウェア工学，システム性能評価，分散データベース，コンピュータグラフィックスなどの研究を行っている．1960年電気学会進歩賞，1968年電子通信学会業績賞，1971年紫綬褒章，1975年情報化個人表彰．計測自動制御学会，日本機械学会等の会員．本学会理事，副会長，計測自動制御学会理事，第3回日米コンピュータ会議議長，第6回ソフトウェア工学国際会議議長等を歴任．1983年より日本ソフトウェア科学会理事長．1985年よりシグマ・システム開発委員会委員長．著書「オンラインリアルタイムシステムの設計」ほか．