ソフトウェア工学 40－12
（1985. 2. 8）

仕様記述法の味見

加藤潤三、染谷誠、板倉教、田端福雄、森沢好臣、山崎利治
日本ユニバック（株）

1．はじめに

Liskov-Zilles［1］以来、プログラムの仕様記述についての議論が盛んである。しかし、仕様の大切さや、記述方法の持つべき特質などを論じたものが多く、どう書いたかといった現場のプログラマにとって具体的な指針となるものが少ないようである。

いい仕様の記述ないしは課題の提示のための考え方や書き方を模索して、そこで学んだ記述の枠組みや記法を小さな課題に適用してみた。理解を確かめるためには実例への適用が一番であると考えたからである。これはその報告である。

2．課題

課題としてボウリング・ゲームの得点計算を選んだ。この課題を選んだ理由は強いていうなら小さいが自明でない程度の複雑さを持っていると考えたからである。規則集［2］より関係のある部分を抜粋する。

（ゲームの構成）

第101条　ボウリングの1ゲームは，10個のフレームをもって構成する。

競技者は，ストライクの場合を除き，それぞれのフレームで2回づつ投球する。ただし，第10フレームがストライクまたはスペアーの場合には，サービスフレームとし，ストライクの場合は，更に2回投球でき，スペアーの場合には1回投球できる。

ゲームの成績は，適正な投球によって，倒されたピンの数をもって計算し，10個のフレームの合計点によって，これを表わす。

投球されたボールとは，競技者の持っているボールが，手から放れ，ファールラインを越えたものを投球されたボールという。

（ストライク）

第102条　第1回の投球によって，完全に配置された10本のピンを全部倒した場合は，これをストライクという。

ストライクの場合は，そのフレームの小さな枠の左に（X）印をつけて表わす。

ストライクの場合，そのフレームのピンの得点は，競技者が更に次のフレームの得点が加算されるので次のフレームの投球を終わるまで空欄としておく。

ストライクに続いて，次のフレームでスペアーの場合には，その得点は20となる。

（ダブル）

第103条　続けて2回ストライクの場合は，これをダブルという。

この場合，第1のストライクのフレームにおける得点は，競技者が更に次の投球を終るまで空欄としておく。

ダブルの場合，最初のストライクの得点は，20にそれに続く第3のフレームの第1球で倒されたピンの数を加える。従って，第2のストライクに続いて次の第1球で9本のピンを倒した場合には，第1ストライクのフレームには29を加える。

（トリプルまたはターキー）

第104条　続けて3回ストライクの場合は，トリプルまたはターキーという。

この場合，第1のストライクのフレームにおける得点は30となる。このようにして10フレーム全部について連続12回のストライクのときは，そのゲームは300の得点となりパーフェクトゲームという。

（スペアー）

第105条　どのフレームにおいても，第2回の投球によってピンの全部を倒した場合は，これをスペアーという。

スペアーの場合は，そのフレームの小さな枠の右に（／）印をつけて，これを表わす。第1回の投球で倒されたピンの数は，スペアーをとる前に小さな枠の左に記入し，そのフレームの得点は競技者が次のフレームで第1回の投球を終るまで空欄とし，次の第1球の得点をスペアーとしての得点10に加え，その合計をもって記入する。第10フレームにおいてスペアーの場合は，サービスフレームとして続いて第3回を投球できる。

（エラー）

第106条　1つのフレームで2回投球し，10本のピンを全部倒すことができなかった場合は，これをエラーという。

エラーの場合には，第1回の投球で倒したピンの数を小さな枠の左に記入し，第2回の投球で残ったピンを1本も倒さなかったとき小さな枠の右に（－）印をつけて表わす。また，第2回の投球で倒したピンの数は小さな枠の右に記入し，そのフレームの得点は第2回の投球が終了すれば直ちに合計し記入する。

（ファール）

第108条　第1回あるいは第2回の投球のとき，ファールをした場合，第1投球のファールは左の枠，第2投球のファールは右の枠の中へ（F）印をつけてこれを表わす。

フレームの第1回の投球でファールをした場合には，倒したピンは全部立て直され，第2回の投球で倒したピンだけをそのフレームで得点として計算する。

もし，第1投でファールをし，第2投で全部倒した場合は，スペアーとなり，第2投で全部倒さなかった場合はエラーとなる。

第2投でファールをおかした場合には，第1投で倒したピンの数だけをそのフレームで計算する。

第10フレームにおいて，第1投でファールをし，第2投で全部のピンを倒してスペアーとなった場合は，サービスフレームとして第3投を投げ，このフレームの計算はスペアーに第3投で倒したピンの数を加えたものとなる。

第10フレームで第3投がファールとなった場合は，最初の2回の投球により倒したピンの数だけを計算する。

（ガター）

第109条　第1回の投球でボールがガターに入った場合は，左の枠の中に(G)印をつけてこれを表わす。

スペアーをとるために投球した第2投がガターに入った場合は，右の枠の中に（－）印をつけてこれを表わす。

ガターの場合は得点は0となるが，第2投により10本のピンを全部倒した場合はスペアーとして計算する。

スコアの記入例

フレーム	1	2	3	4	5	6	7	8	9	10
	X	X	X	⑧/	F 8	9 F	X	8 －	G 9	X X 8
得　点	30	58	78	88	96	105	123	131	140	168

3．仕様の記述

ボウリング・ゲームの得点計算の仕様を記述したい。ここで「仕様」というのは、「課題」、「算法」、「プログラム」を漠然と区別して、「課題の提示」ないしは「課題の背景」「課題のモデル化」といった程度の意味である。さて我々は課題の提示として次のような記述方法を試みた。

（1）述語論理の言葉によって：
　　ＩＰＬ approaches ［ 3］
（2）文法記述にならって：
　　ＤＣＧ［ 4］、属性文法［ 5］、
　　表示的意味記述［ 6］、ＶＤＭ［ 7］
（3）抽象データ型によって：
　　ＨＤＭ［ 8］、ＨＩＳＰ［ 9］
（4）並列処理的に：
　　Concurrent Prolog ［10］
（5）一般的：
　　染谷［11］、Ｚ［12］

我々はゲーム体験から、得点計算を手続き的に理解していた。それをそのまま書けば例えばPascalプログラムが完成する。そうではなくて書くべきものは「仕様」であり、また記法として既出のようなものを考えていたから、それらによく合致するような課題の理解の仕方を考えることになった。そこでゲームを大略つぎのように理解しようとした。

（1）ほとんどスクラッチから集合、関係を述語論理の言葉のよって構成しながら（抽象）ゲームを記述する。
（2）文字列である入力データに注目し、課題記述を文法記述にならい、その構文的側面と意味的側面をほどよく調和させることを工夫する。
（3）抽象データ型として課題を把握するためにひそかに状態機械を考える。
（4）勝手に集合や関数（逆関数）を定義して（抽象）ゲームを構成する。スコアはゲームの「表現」と考え、表現に対しゲームが一意に存在する（あるいは存在しない）ことを注意して「仕様」を完成する。

以下個別に記述を概観する。

3.1　ＩＰＬ approaches

ＩＰＬapproachesでは、entitiesとpropositionsによって課題の世界をモデル化する。ボウリング・ゲームでは、Game,Frame,Throw,...をentitiesとして導入し、投球ルール、スコアリング・ルール、..をこれらの間に成り立つpropositionsとして与える。”集合”と”関係”という単純な概念によってのみ、モデル世界を構築できる。しかしその反面、記述（思考）のための道具が原始的なものしかないため記述量が増えた。

実現のための手掛かりをえるためには、記法の洗練、より簡便な組込み述語の導入などが考えられる。次にＩＰＬapproachesによる仕様の一部を示す。

```
For all T For all t For all P For all G
                          (G throws TtP Iff Prb'''GTtP)
For all T For all t For all P For all G
    (If G Throws Ttp Then t Is among Nn
                      & P Is among Bowlingpoint)
Games = {G/For some T for some t for some P (G throws TtP)}
Throws={T/For some G for some t for some P (G throws TtP)}
Games In I
Throws In I
Throw Number = {tT/For some G For some P (G throws TtP)}
Throw Number Is Among Fcn
Dr Throw Number = Throws
For all G Among Games
    ({t/G throws TtP) = {1,2,···,K : {T/G throws TtP}))
 .
 .
 .
For all G For all F For all f (G frame Ff Iff Prc'''GFf)
For all G For all F For all f
    (If G frames Ff Then G Is among Games & f Is among Nn)
Frames = {F/For some G For some f (G frame Ff)}
Frames In I
Frame Number = {fF/For some G (G frames Ff)}
Frame Number Is among Fcn
Dr Frame Number = Frames
For all G Among Games
    ({f/G frame Ff) = {1,2,···,10})
 .
 .
 .
For all T Among Throws For all F Among Frames
    (If T is first throw of F
     Then Throw Point : T is among digit U {strike})
For all T Among Throws For all F Among Frames
    (If T is second throw of F
     Then Throw Point : T is among digit U {spare})
For all T Among Throws For all F Among Frames
    (If T is first throw of F & Throw Point : T = Strike
     Then K : {T/T is second throw of F) = 0)
```

3.2　属性文法

記述の様式はKnuth ［ 5］をそのまま真似ている。構文部分を次の程度にとどめる。

```
bgame  -> body
body   -> frame body | frame
frame  -> throw
        | throw throw
        | throw throw throw
throw  -> 'G' | 'F' | '-' | 'X' | '/'
        | '1' | '2' | '3' | '4' | '5'
        | '6' | '7' | '8' | '9'
```

上の構文範ちゅうに下のような属性を付随させる。

symbols	attributes		
	inherited	synthesized	domains
bgame		score	{0,...,300}
body	no		{1,...,10}
		score	
		first	{0,...,10}
		second	{0,...,10}
frame	no		
		type	{error,spare,strike}
		first	
		score	
		second	
throw	no		
	order		{1,2,3}
		score	
		type	

以上によって、

条件、生成規則、相続属性、合成属性の形式で意味部分を補足する。このとき、bodyの合成属性としてbody内第一投first 、第二投secondを用意したので、実際のゲームに際しての得点計算と同様の計算規則の提示になっている。

3.3 DCG

3.2 の属性文法の場合と同様のアイデアをDCG処理を備えたPrologによって課題を記述する。具体的にはPrologプログラムである。一部を次に示す。最大の特長は「実行可能である」ことである。

```
bowling(X) :- bowling_score(Total,X,[]),
              write('Total Score is '),
              write(Total),nl.
bowling_score(Total) --> frames(Total,1,_,_).
frames(T,FN,FTV,STV)
  --> { FN<10 },error_frame(FTV,STV),
      { Next_FN is FN+1 }, !,
      frames(Next_T,Next_FN,Next_FTV,Next_STV),
      { T is Next_T+FTV+STV }.
frames(T,FN,FTV,STV)
  --> { FN<10 },spare_frame(FTV,STV),
      { Next_FN is FN+1 }, !,
      frames(Next_T,Next_FN,Next_FTV,Next_STV),
      { T is Next_T+FTV+STV+Next_FTV }.
frames(T,FN,FTV,STV)
  --> { FN<10 },strike_frame(FTV),
      { Next_FN is FN+1 }, !,
      frames(Next_T,Next_FN,Next_FTV,Next_STV),
      { STV is Next_FTV,
        T is Next_T+FTV+Next_FTV+Next_STV }.
frames(T,FN,FTV,STV)
  --> { FN=10 },ten_frame(FTV,STV,TTV),
      { T is FTV+STV+TTV }.
error_frame(FTV,STV) --> error_1st_throw(FTV),
                         error_2nd_throw(STV),
                         { FTV+STV < 10 }.
spare_frame(FTV,STV) --> error_1st_throw(FTV),
                         spare,
                         { STV is 10 - FTV }.
strike_frame(FTV) --> strike,{ FTV is 10 }.
error_1st_throw(FTV) --> ("G",!,{ FTV is 0 });
                         ("F",!,{ FTV is 0 });
                         digit(FTV).
error_2nd_throw(STV) --> ("-",!,{ STV is 0 });
                         ("F",!,{ STV is 0 });
                         digit(STV).
```

3.4 表示的意味記述

これは直接的な意味記述（direct semantics）、すなわち状態推移のみによるものをOxford流で行なったものである。一部を次に示す。ここでは直接的意味記述によりたかったため、モデルとしてはやや人為的な変形を加えたものになっている。

SYNTACTIC DOMAINS

$\Sigma \in Sc$ (Scores)
$\Phi \in Bd$ (Bodies)
$\Psi \in Fr$ (Frames)
$T \in Th$ (Throws)

SYNTAX

$\Sigma ::= \Phi$
$\Phi ::= \Psi\Phi \mid \Psi$
$\Psi ::= T \mid TT \mid TTT$
$T ::= $ 'X' | '/' | 'G' | '-' | 'F'
| '1' | '2' | '3' | '4' | '5'
| '6' | '7' | '8' | '9'

SEMANTIC DOMAINS

Z (Integers)
$Ty = \{error, spare, strike, double\}$ (Types)
$No = \{1, 2, \ldots, 10\} \in Z$ (Frame Nos)
$Pt = \{0, 1, \ldots, 300\} \in Z$ (Points)
$\sigma \in S = No \times Ty \times Pt$ (States)

SEMANTIC FUNCTION

$S : Sc \rightarrow Pt$
$B : Bd \rightarrow S \rightarrow S$
$F : Fr \rightarrow S \rightarrow S$
$N : Th \rightarrow Z$

3.5 VDM

これは3.3 をVDMの記述言語MetaIVによって書いたものである。その一部を次に示す。

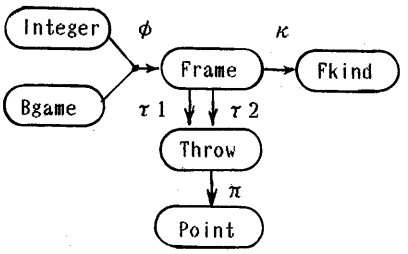
```
Syntactic Domains
 Game :: Properframes [Serviceframe]
 Properframes = Properframe+
 Serviceframe = Onemorethrow | Twomorethrow
 Properframe = Strike | Spare | Error
 Onemorethrow = STRIKECHAR | (Errorchar - {G})
 Twomorethrow :: Onemorethrow (Onemorethrow | SPARECHAR )
 Error :: Errorchar Errorchar
 Spare :: Errorchar SPARECHAR
 Strike :: STRIKECHAR
 Errorchar = G | - | F | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Context conditions
 is-wf-Game : Game -> Bool
 is-wf-Game (mk-Game(pf,sf)) ≜
  len(pf) = 10
  ∧(∀i∈{1:10})(cases fl[i]
                 (mk-Error(d1,d2) -> int(d1)+int(d2)<10∧d1≠-∧d2≠G,
                  mk-Spare(d,s) -> d≠-,
                  mk-Strike(x) -> TRUE,
                  T -> FALSE)
  ∧(cases sf
     (NIL -> is-Error(pf[10]),
      mk-Onemorethrow(t) -> is-Spare(pf[10]),
      mk-Twomorethrow(t1,t2) ->
            is-Strike(pf[10])
          ∧(t1=STRIKECHAR => t2∈Onemorethrow)
          ∧(t1≠ STRIKECHAR ∧t2=STRIKECHAR =>
              (t2=SPARECHAR ∧int(s1)+int(s2)<10)∨t2=SPARECHAR),
      T -> FALSE),T -> FALSE)
```

3.6 HDM

HDMは抽象データ型によるモジュール作成用言語とのことであるが、o，v関数などによって相当状態機械を意識させられるものである。表現の歪（過剰仕様）がでる。

3.7 HISP

染谷（3.9 後出）の仕様を抽象データ型として書いたものであり、種（sort）とそれらの間の演算子をGougen図［13］で次に示す。



3.8 Concurrent Prolog

構文部分の記述throwsと意味部分の記述scoresに分け、それぞれをプロセスとみなし、ゲームの各フレーム結果の流れをプロセス間通信とみなした。一部を次に示す。

```
bowling(X) :- throws(Ps,X?,[]),
              scores(Ps?,0,GT ),
              total_score(GT?).
throws(Ps,S0,S) :- frames(1,Ps,S0?,S).
frames(FN,[[FN,error,FTV,STV]|Ps],S0,S)
     :- FN<10,error_frame(FTV,STV,S0,S1) |
              Next_FN is FN+1,
              frames(Next_FN,Ps,S1?,S).
frames(FN,[[FN,spare,FTV,STV]|Ps],S0,S)
     :- FN<10,spare_frame(FTV,STV,S0,S1) |
              Next_FN is FN+1,
              frames(Next_FN,Ps,S1?,S).
frames(FN,[[FN,strike,FTV]|Ps],S0,S)
     :- FN<10,strike_frame(FTV,S0,S1) |
              Next_FN is FN+1,
              frames(Next_FN,Ps,S1?,S).
frames(FN,[[FN,ten,FTV,STV,TTV]],S0,S)
     :- FN=10 | ten_frame(FTV,STV,TTV,S0,S).
scores([[FN,strike,FTV]|Ps],T,GT)
     :- FN =< 8,wait(Ps),Ps=[P1,P2|_],
        P1=[_,strike,STV],wait(P2),
        P2=[_,_,TTV|_],wait(TTV) |
        New_T is T+FTV+STV+TTV,
        scores(Ps,New_T,GT).
scores([[FN,strike,FTV]|Ps],T,GT)
     :- wait(Ps),Ps=[[_,_,STV,TTV|_]|_],
        wait(TTV) |
        New_T is T+FTV+STV+TTV,
        scores(Ps,New_T,GT).
scores([[FN,spare,FTV,STV]|Ps],T,GT)
     :- wait(Ps),Ps=[[_,_,TTV|_]|_],wait(TTV) |
        New_T is T+FTV+STV+TTV,
        scores(Ps,New_T,GT).
scores([[FN,error,FTV,STV]|Ps],T,GT)
     :- wait( STV ) |
        New_T is T+FTV+STV,
        scores(Ps?,New_T,GT).
scores([[FN,ten,FTV,STV,TTV]],T,GT)
     :- wait( TTV ) | GT is T+FTV+STV+TTV.
total_score(GT) :- wait(GT,T) |
        write('Total Score is '),write(T),nl.
```

3.9 染谷

抽象ボウリング・ゲームの母体（枠組）を集合Bgame,Frame,Throw,Point,Fkind とそれらの間の演算π（投球による得点）、τ1,τ2（フレームにおける第一投、第二投）、κ（フレーム型すなわち、ストライク、スペア、エラー）、φ（ボウリング・ゲームのスコアフレーム分け）によって定義する。これらの間に成立する公理を羅列する。ついでフレーム得点をゲームに対して、フレームから得点への関数を対応させる関数として定義する。さらにゲーム得点をフレーム得点を用いてゲームから得点への関数として定義したものである。

```
1．枠組
 .0  <Bgame,Frame,Throw,Point,Fkind, π, τ1,τ2,κ, φ>
 .   is BowlingGame <=>def
 .1    Bgame ≠ φ
 .2  ∧Frame ≠ φ
 .3  ∧Throw ≠ φ
 .4  ∧Point = NaturalNumber
 .5  ∧Fkind = {error,spare,strike,service1,service2}
 .6.0  ∧π :Throw→Point
  .1   ∧(∀t∈Throw)(0≦π(t) ≦10)
 .7.0  ∧τ1:Frame →Throw
  .1   ∧τ2:Frame →Throw ∪{undefined}
  .2   ∧κ  :Frame →Fkind
  .3   ∧(∀f∈Frame)
  .4     {[ κ(f)=error => (τ2(f)∈Throw
  .5                      ∧π (τ1(f)) + π (τ2(f)) < 10)]
  .6    ∧[ κ(f)=spare => (τ2(f)∈Throw
  .7                      ∧π (τ1(f)) + π (τ2(f)) =10)]
  .8    ∧[ κ(f)=strike => (τ2 =undefined
  .9                       ∧π (τ1(f)) =10)]
  .10   ∧[ κ(f)=service1 => τ2(f)=undefined ]
  .11   ∧[ κ(f)=service2 => [ τ2(f)∉Throw
  .12       ∧(π (τ1(f))<10 => π (τ1(f)) + π (τ2(f)) ≦10 )]]
  .13   }
 .8.0  ∧φ:Bgame→Frame   /* the set of sequences in Frame */
  .1   ∧(∀b∈Bgame)
  .2     {[ len(φ(b))=10 ∨ len(φ(b))=11 ]
  .3    ∧(∀i∈{1:10})  [κ (φ(b)(i)) ∉ {error,spare,strike}]
  .4    ∧[ κ (φ(b)(10))=error => len(φ(b))=10 ]
  .5    ∧[ κ (φ(b)(10))=spare => len(φ(b))=11
  .6                             ∧κ (φ(b)(11))=service1]
  .7    ∧[ κ (φ(b)(10))=strike => len (φ(b))=11
  .8                              ∧κ (φ(b)(11))=service2]
  .9    }


2．得点
 2.1  frame-point:Bgame→[{1:10} →Point]
  .0  frame-point(b)(i) =def
  .          /* defined by conditional expression */
  .1     [ κ (φ(b)(i))=error →π (τ1(φ(b)(i))) +π (τ2(φ(b)(i)));
  .2       κ (φ(b)(i))=spare →10 +π (τ1(φ(b)(i+1)));
  .3       κ( φ(b)(i))=strike→
  .4          [ τ2(φ(b)(i+1)) ∈ Throw→
  .5               10 +π (τ1(φ(b)(i+1))) +π (τ2(φ(b)(i+1)));
  .6            τ2(φ(b)(i+1)) = undefined →
  .7               10 +π (τ1(φ(b)(i+1))) +π (τ1(φ(b)(i+2)))
  .8    ]    ]

 2.2  score:Bgame→Point
  .0    score(b)=def Σ1≦i≦10frame-point(b)(i)
```

3.10 Z

これは3.9 をＺ言語で書き直したものである。その一部を次に示す。

```
BOWLING-POINT =
 def
  strike = {'X'}
  spare = {'/'}
  digit = {'0','1','2','3','4','5','6','7','8','9'}
  bowlingpoint = {strike,spare} U digit
  int = {'0'→0,'1'→1,'2'→2,'3'→3,'4'→4,'5'→5,
         '6'→6,'7'→7,'8'→8,'9'→9}
  frame = {1,2,3,4,5,6,7,8,9,10}
  throw = set f for f : NAT where f∈segment[NAT] end
 end BOWLINNG-POINT

BOWLING-GAME =
 use BOWLING-POINT def
  throw-rule = class
                φ : throw → frame;
                τ : throw → bowlingpoint
               end;
  frame-order = subclass
                 throw-rule where forall t for t : throw
                 then
                  (t+1)∈throw ⇒ φ(t+1) = φ(t)
                                or φ(t+1) = φ(t) + 1 end
                end;
  frame-rule =
     subclass throw-rule where
       forall f for f : frame - {10} then
        exist t for t : throw where
         inverse(φ)(f) = {t} and τ(t) = strike
         or
         inverse (φ)(f) = {t,t+1} and τ(t)∈digit and
         [[τ(t+1)∈digit and int(τ(t))+int(τ(t)) < 10]
           or τ(t+1) = spare]
        end
       end
```

4．味見と仕様記述の比較

小さな課題についていろいろ仕様を書きまた読んでみた。当然ながら課題の認識とその記述は書き手の主観と記述に使った規範に依存し、読み手の趣味もからんでくる。このような書き手の主観とか読み手の趣味を割り引いて"書きやすさとか読みやすさ"を主眼にして味見した結果を述べる。

（１）予備知識が不用なものは書きやすく、読みやすい。いい例が染谷（3.9 ）である。

（２）文法記述の方法は表層構造（構文）から深層構造（意味）への関数を定義するものであったが、逆に深層構造の表現関数によって表層構造を定義することも可能である。これが結構わかりやすく、また書きやすいものになる。これは深層構造が内在的（intrinsic ）な性質だけに注目して議論を展開できることに起因するためと考えられる。

（３）記述言語の背景にあるものの見方いわば基本原理がわかりやすいと、書きやすく、読みやすくなる。例えばＨＩＳＰがそうである。

（４）構造的プログラミングの精神すなわち抽象化と分割はわかりやすさの根本である。課題の分割の強力な原理として並行性（Parallelism ）を挙げることができる。試みてはいないが算体主導型もそうであろう。

（５）特定の記述言語を用いる場合と、そうでない場合では、書き手の能力に強く依存して相違が生ずる。すなわち、よく書けていればいいし、へたをすると読めないものになる。記述言語が一般的であればあるほど書き手に左右され、読みやすい仕様になったり、ならなかったりすることになる。言語を固定することは、実は言語が思考の道具として働くので課題の認識の手掛かりを与えてくれるから一面役立つ。しかし、記法が課題にとって有力な概念を支援していないと、仕様に表現の歪が生じ過剰仕様になりがちである（例、ＨＤＭ）。固定した言語が簡素さを尊重して原始的な概念とわずかな構成法だけを与える場合、実際的でない面が生じる。（例、ＩＰＬ approaches ）。

（６）「自由に書く」ことは多種論理（Many-sorted Logic）や、高階論理を勝手に使えることを意味し、特に高階の関数などが有効であるようだ。したがって記述言語がこのような概念を許容するかどうかが記述能力とともに、わかりやすさの根元に影響する。

なお、つぎのような項目についても議論した。

整合性の検査や虫とりのしやすさ

仕様がある程度の量の記述を伴うとき、書いた仕様の虫とりや整合性の検査がプログラムと同様やはり問題になる。その面でＤＣＧ／Prolog、ＨＩＳＰなどの優位は明らかである。整合性の検査として強力な手段は型検査であろうから、ＬＣＦ／ＭＬなどのような多様型（Polymorphic type）を許す記述言語（プログラム言語）が好ましいものになる。

実現への手掛かり

文法記述にならった場合は、コンパイラの作成の要領の実現手段を示唆している。染谷は一見実現手段を隠しているようであるが、よく眺ると表現関数の逆関数を意味関数とみなすことによって実現の手掛かりがみえてくる。ＩＰＬによるものもデータベース構築のためのものであると考えれば、データベースに付随する手続き（意味ルーチン）を作成すればいいといった実現へのヒントが得られることになる。ところで、課題を構成的数学の枠組みで定理として述べその証明にしたがってプログラムを作成（自動作成）することは美しいやり方である。

適用範囲

記法の適用範囲はそれが一般的かつ強力な概念を含有しているかどうかによって定まる。したがって勝手流に書いた染谷が一番普遍的である。これから他の記法への翻訳が容易であった。

５．まとめ

算体主導型など他にも見るべき規範は多い。残念ながら、まだこれらによる記述を試みていない。

あたりまえであるが、課題の持つ"構造"はその理解者の認識に依存する。しかし、その構造には幾つかの"型"があり、それらを下敷にして課題を眺めることができるようである。ある種の句構造をみいだすとき文法記述にならって課題を提示するころができる。しゃにむに抽象データ型として提示することも勉強になる。前件と後件を述語論理の言葉で書くこともいい。これらの多くがいずれも実現への手掛かりを与えてくれることも特筆できる。

なんといっても一番一般的な記述方法は数学理論の展開を見習うことである。わかりやすくまた厳密な課題提示にとって、数学理論がもつ抽象性、論理性、普遍性、形式性といった特性はきわめて好ましい。課題を枠組みや書式にとらわれることなく、自由にかつ数学的に示すことがいい仕様記述につながることになる。このような精神と態度で、課題に応じてまた読み手を考えて書き方を工夫することが仕様の記述にとってもっとも大切である。このあたりまえのつまらない事実が結論である。

参考文献

[1] Liskov,B. and Zilles,S.: An Introduction to Formal Specifications of Data Abstractions,Current Trends in Programming Methodology, Vol.1, Prentice-Hall(1977).

[2] 全日本ボウリング協会：ボウリング規定集、(財) 全日本ボウリング協会 (1981) .

[3] van Griethuysen,J.J.(ed): CONCEPTS and TERMINOLOGY for the CONCEPTUAL SCHEMA and the INFORMATION BASE,ISO/TC97/SC5-N695(1982)

[4] Pereira,F. and Warren,D.:Definite Clause Grammars for Language Analysis, Artif. Intell., Vol.13, pp.231-278(1980).

[5] Knuth,D.E.:Examples of Formal Semantics, Lecture Notes in Mathematics, No.188, pp.212-235, Springer-Verlag(1971).

[6] 中島玲二：数理情報学入門，朝倉書店(1982)

[7] Bjorner,D and Jones,C.B.(eds):The Vienna development Method,Lecture Notes on Computer Science, No.61, Springer-Verlag(1978).

[8] Silverberg,B.A. et al.: The HDM Handbook 3 vols, SRI International(1979).

[9] Futatsugi,K. and Okada,K.:A Hierarchical Structuring Method for Functional Software Systems,6th International Conference on Software Engineering,pp.393-402,IEEE(1982).

[10] Shapiro,E.Y.:A Subset of Concurrent Prolog and Its Interpreter, ICOT TR-003 (1983).

[11] 日本ユニバック㈱：RSDMプログラム仕様記述技法，日本ユニバック㈱(1983).

[12] Abrial,J.R.,Schuman,S.A. and Meyer,B.: Specification language(Draft),Proceedings of Summer School on Program Construction, Cambridge University Press(1979).

[13] Goguen,J.A. et al.: An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types, Current Trends in Programming Methodology , Vol.4, Prentice-Hall(1978).