イオタ入門

湯浅太一，小島啓二，柴山悦哉，萩野達也，中島玲二（京大・数理研），本田道夫（香川大・経済）

はじめに

データ抽象化（data abstraction）や手続き抽象化（procedural abstraction）にもとづいた階層的でモデュラーなプログラム作成が，信頼性が高く維持管理の容易なソフトウエアの作成やプログラム検証の困難の克服に役立つことは広く知られている。この方法によるプログラム作成に適した環境を与えるための，総合的な対話的プログラミング支援システムとしてイオタ・システムが開発された。（イオタ・システムの考え方と方法については[2,4]を参照されたい。）本講演では，イオタ・システムにおいて，プログラム・モデュールが開発される過程を，（プログラム検証も含めて）デモンストレーションを通じて解説する。以下にイオタ・システムが基礎とする，プログラムと形式的仕様記述のための言語，イオタ語について概説し，イオタ・システムの概観を示し，最後にイオタ語でかかれたプログラムの例をあげる。

1. イオタ語

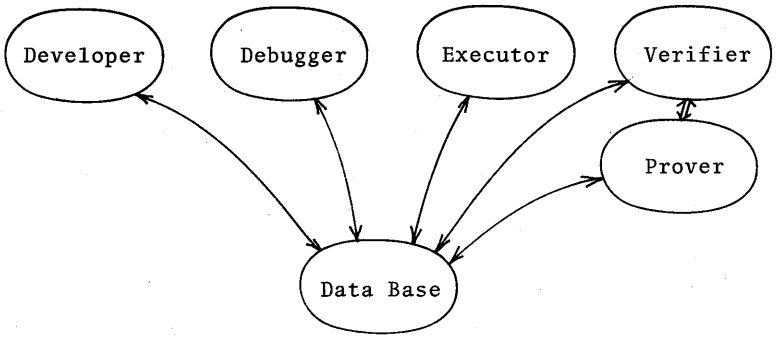
イオタ語とその理論的基礎及び検証方法等の詳細については[3]を参照されたい。本節では，以下のプログラム例を理解するのに必要なイオタ語の基本的構造を概説する。

イオタ語でかかれるプログラムは，階層的構造を成すプログラム・モデュールから構成される。各モデュールは，そのモデュールで定義される関数の宣言などからなるインタフェイス部（interface part），それらの形式的仕様を与える仕様部（specification part），及びプログラム実現を記述する実現部（realization part）から成る。このうち，インタフェイス部と仕様部は，モデュールの抽象的記述を行なうところから，抽象部分（abstract part）とよばれ，実現部と明確に区別されている。あるモデュールMが他のモデュールNを使って作成されているとき，MはNに依存しているという。ここで，MはNの抽象部分のみに依存し，Nの実現部分には依存しない。

モデュールはその役割に応じて，type，sype，procedure の3種類に分類される。type モデュールは抽象データ型を定義し，procedure モデュールでは即に定義されたデータ型を使った手続きが導入される。sype モデュールは，いくつかのデータ型から成る族（class）を定義する。（sype については後出のプログラム例の中で具体的に述べる。）

2. イオタ・システム概観

イオタ・システムは，階層的でモデュラーなプログラム作成に必要なすべての情報をもったDataBaseを中心として，Developer，Debugger，Executor，Verifier，及びProver の5つのサブ・システムから構成される。（下図で二重線はコントロールの流れ，実線はデータの流れをあらわす。）



プログラム・モデュールはDeveloperとの対話により入力，変更され，直ちに解析が行なわれ，その過程で得られた様々な情報とともに，内部形式の形でData Baseに保存される。モデュールの対話的入力，変更を効率よく行なうために，Developerではプログラム解析とエディティングの機構を有機的に結びつけたスキーマがとられている([5]参照)。

Debuggerは，Data Base内の内部形式プログラムを直接実行し，tracerやbreakerなどの機能によって動的なデバグを支援する。

Executorは内部形式プログラムをオブジェクト・プログラムに変換し，実行するものである。

Verifierは各モデュールの仕様が実現部で正しく実現されていることを検証するために必要な情報の管理を行ない，また検証に際して証明されなければならない論理式(formula)を作成してProverにうけわたす。Proverはmany-sortedな一階述語論理のための対話的定理証明システムであり，特に階層的でモデュラーな構造をもった理論の上の演繹を能率的に行なうように設計されている[1]。

3. プログラム例

以下にイオタ語で書かれた簡単なプログラム・モデュールの例をいくつかあげる。

Fig.1は整数のデータ型を定義するtypeモデュールである。ここで"@"はこのモデュールで定義されるデータ型，即ちここではintをあらわす。'AS'によって関数名の関数記号による省略が導入される。(例えば plus(x,y)と書くかわりにx+yと書いてもよい。)

今，2つの引数をうけとり，その'小さい方'及び'大きい方'をそれぞれ値としてかえす関数minとmaxを作りたいとしよう。これらの関数の引数が属するデータ型は，元の間に全順序関係が定義されていれば何でもよいわけである。そこで全順序関係の定義されたデータ型をあらわすtypeモデュールorderを導入する。(Fig.2)

あるデータ型Tに全順序関係があるとき，即ちTに関数lesseqが定義されてい

て、しかもそれがorderの仕様部の条件をみたすときに、order≦Tと書く。例えばorder≦intが成り立つ。

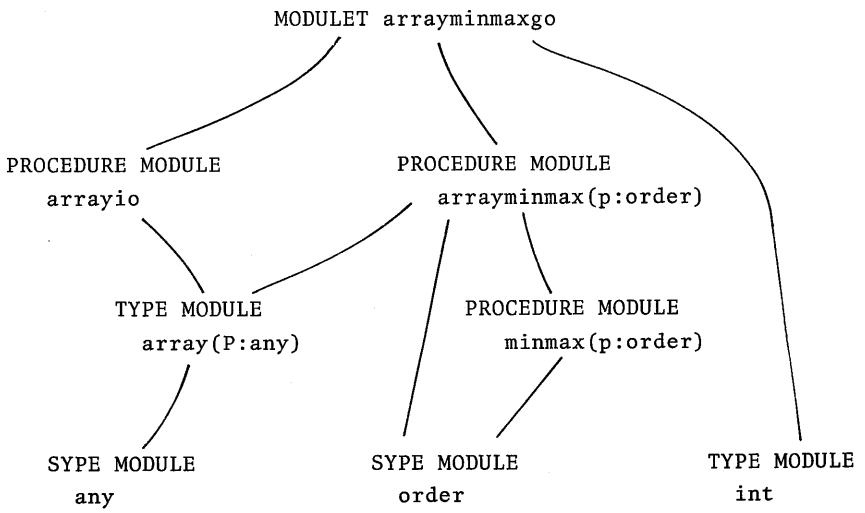
orderを使って タイプ・パラメータ化されたprocedureモデュール minmaxを作成することができる。(Fig.3) ここで pはタイプ・パラメータで，order≦Tなる条件をみたすデータ型に束縛される。例えば pがintに束縛されれば、minは2つの整数をうけとり，その小さい方の整数を値としてかえすと解釈される。実現部にはminとmaxの実現関数 ↓minと ↓maxの bodyが与えられている。IF文の条件式において pがあるデータ型Tに束縛されたときは、そのデータ型の上で定義された関数 lessegがよばれる。

Fig.4は配列 (array)を定義するtypeモデュールである。配列の要素の属するデータ型は何でもかまわないので，このモデュールはあらゆるデータ型をあらわすsype 'any'を使ってタイプ・パラメータ化されている。ここで定義されている関数のうち store, shrink, stretch は operationとよばれ，引数のうち、interface 部の宣言の'!'の左側にあらわれる引数は call by variableでうけわたされる。なお、このモデュールは上の int同様、言語に基本的なデータ型として、前もって定義されたものであり、 store(x!i,a)のかわりに x[i]:=aと書くことも許されている。

procedureモデュール arrayminmax では、order≦Tなるデータ型Tを要素のデータ型とする配列をうけとり、その要素のうち最小と最大のものをそれぞれ値としてかえす関数が定義されている。

これらのモデュールの実行をコントロールするためのプログラムがmoduletである。Fig.7で、intarrayread(Fig.6)によって整数を要素とする配列を入力し、その中の最大の要素と最小の要素を出力するための moduletarrayminmaxgoを示す。

以上のモデュール及びmoduletが構成する階層を次に図示する。(モデュール間の実線は依存関係をあらわす。)



参考文献

1. Honda, M., Nakajima, R.: An Interactive theorem proving system for structured theories, IJCAI '79, pp. 400～402.
2. Nakajima, R., Yuasa, T., Kojima, K.: The 2 programming system — a support system for hierarchical and modular programming —, IFIP '80 pp. 229-304
3. Nakajima, R., Honda, M., Nakahara, H.: Hierarchical program specification and verification — a many-sorted logical approach, Acta Informatica 14, 135—155 (1980)
4. 湯淺太一，中島玲二：イオタ語の言語処理およびプログラム作成支援システム；第21回プログラミング・シンポジウム，1980
5. Yuasa, T.: Design and implementation of 2 language system, (to appear)

```
INTERFACE TYPE int
    FN  zero:  → @    AS 0
        one:   → @    AS 1
        neg:  @ → @    AS -@
       plus: (@,@) → @    AS @+@
     lesseq: (@,@) → bool    AS @≤@
END INTERFACE

SPECIFICATION TYPE int
    VAR x,y,z:@;
    AXIOM  1: x+y=y+x
           2: x+(y+z)=x+y+z
            ....
           8: x≤y ∧ y≤z ⊃ x≤z
           9: x≤y ∨ y≤x
          10: x≤y ∧ y≤x ⊃ x=y
            ....
          13: x≤x
END SPECIFICATION
```

Fig. 1 type module int

```
INTERFACE SYPE order
    FN lesseq : (@,@) → bool   AS @≤@
END INTERFACE

SPECIFICATION SYPE order
    VAR x,y,z:@;
    AXIOM 1: x≤x
          2: x≤y ∧ y≤z ⊃ x≤z
          3: x≤y ∧ y≤x ⊃ x=y
          4: x≤y ∨ y≤x
END SPECIFICATION
```

Fig. 2 sype module order

```
INTERFACE PROCEDURE minmax (p: order)
    FN min : (p,p) → p
       max : (p,p) → p
END INTERFACE

SPECIFICATION PROCEDURE minmax (p:order)
     VAR x,y:p;
     AXIOM 1: x≤y ⊃ min(x,y)=x
           2: y≤x ⊃ min(x,y)=y
           3: x≤y ⊃ max(x,y)=y
           4: y≤x ⊃ max(x,y)=x
END SPECIFICATION

REALIZATION PROCEDURE minmax(p:order)
    FN ↓min (x,y:p) RETURN (z:p)
       IF lesseq(x,y) THEN z:=x
                      ELSE z:=y
       END IF
    END FN
    FN ↓max (x,y:p) RETURN (z:p)
       IF lesseq(x,y) THEN z:=y
                      ELSE z:=x
       END IF
    END FN
END REALIZATION
```

Fig. 3 procedure module minmax

```
INTERFACE TYPE array(p:any)
    FN  create: (int,p) → @
        high: @ → int
        fetch: (@,int) → p
    OP  store: (@|int,p)
        shrink: (@|)
        stretch: (@|p)
END INTERFACE
```

Fig. 4 type module array

```
INTERFACE PROCEDURE arrayminmax (p:order)
    FN arraymin : array(p) → p
       arraymax : array(p) → p
END INTERFACE

SPECIFICATION PROCEDURE arrayminmax (p:order)
    VAR x:array(p);i:int;
    AXIOM 1: C≤i ∧ i≤high(x) ⊃ arraymin(x)≤x[i]
          2: ∃i(C≤i ∧ i≤high(x) ∧ arraymin(x)=x[i])
          3: C≤i ∧ i≤high(x) ⊃ x[i]≤arraymax(x)
          4: ∃i(C≤i ∧ i≤high(x) ∧ arraymax(x)=x[i])
END SPECIFICATION

REALIZATION PROCEDURE arrayminmax (p:order)
    FN ↓arraymin (a:array(p)) RETURN (z:p)
       IF high(a)=C THEN z:=a[C]
                    ELSE z:=a[high(a)];
                         shrink(a|);
                         z:=min(z,↓arraymin(a))
       END IF
    END FN
    FN ↓arraymax (a:array(p)) RETURN (z:p)
       IF high(a)=C THEN z:=a[C]
                    ELSE z:=a[high(a)];
                         shrink(a|);
                         z:=max(z,↓arraymax(a))
       END IF
    END FN
END REALIZATION
```

Fig. 5 procedure module arrayminmax

```
INTERFACE PROCEDURE arrayio
    FN intarrayread : → array(int)
ENDINTERFACE
```

Fig. 6 procedure module arrayio

```
MODULET arrayminmaxgo
  VAR a:array(int);
  a:=intarrayread;
  SWRITE("
   max=");
  IWRITE(arraymax(a));
  SWRITE("
   min=");
  IWRITE(arraymin(a))
END MODULET
```

Fig. 7 modulet arrayminmaxgo