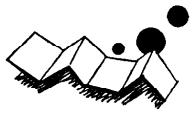
解　説



Ada 実現に関する問題点について†

—ACM コンパイラ作成会議より—

徳　田　雄　洋††

1. はじめに

本稿は，米国国防総省 (U. S. Department of Defense) が 1975 年1月以来進めてきた高水準言語 (High Order Language) プロジェクトの成果である，プログラム言語 Ada* について，設計関係者が実現上の問題を論じた3つの講演から，若干の話題を選んで報告するものである．

本稿の出典となる講演は，1979 年8月 ACM Compiler Construction Conference で特別講演として行われたもので，講師は講演順に Ben Brosgol[1] (Intermetrics 社), Harald Ganzinger[10] (Technical University Munich), Olivier Roubine[15] (Cii-Honeywell-Bull 社) である．また，Ada 実現上の問題を論じた講演ではないが，John Buxton (Warwick 大学) による Ada と PEBBLEMAN[6] 一般に関する 1979 年8月の California 大学 Santa Cruz 校における講演[2]も参考とした．

本稿では，まず Ada の今までの歴史と 1979 年8月現在の状況を，講演で触れた範囲内で整理する．次に，Ada の特徴の中でも極めて厳しい批判に遭遇している overloading (オーバローディング) について，その内容の紹介と実現法の報告を行う．そして最後に，多重タスキングのための新しい特徴である rendezvous (ランデブ) について，内容の紹介および実現法の報告を行う．なお，講演で取り上げられたが，本稿では扱わなかった問題については，付録に問題一覧として整理した．

本稿は，Ada 言語仕様の知識を特に前提とせずに理解できることを目標に作成されているが，詳細を知りたい方のために，いくつかの用語には，Ada 参照マニュアル (Man)[7] または Ada デザイン釈明書 (Rat)[12] の関連セクション番号を付記した．

なお，米国国防総省の高水準言語プロジェクトについては，1977 年までの動向に関して上條氏の報告[13]が，1978 年までの動向に関して安村氏の報告[16]が，また 1979 年春までの動向に関して大野氏の報告[14]があり，Ada のプログラム言語としての輪郭を大体つかむためにも，また開発の背景を知る上にも有用である．また，Ada の最近の情報を伝えるものとしては，高水準言語作業グループ (High Order Language Working Group) により，1979 年5月から "Ada Test and Evaluation Newsletter"[8),9)] が発行されている．

2. Ada 現況

Ben Brosgol の講演[1]と John Buxton の講演[2]の範囲内で，高水準言語プロジェクトの歴史と現状を整理してみよう．

米国国防総省は，1975 年1月に高水準言語作業グループを発足させ，あるべき共通高水準言語の姿を要求仕様の形でまとめる作業を開始した．これらの要求仕様は次々と改訂され，STRAWMAN, WOODENMAN, TINMAN (1976 年6月), IRONMAN (1977 年1月), Revised IRONMAN (1977 年7月), STEELMAN[5] (1978 年6月) としてそれぞれ発表された．

これらの作業と並行して，1976 年に TINMAN を基準とした，既存プログラム言語 23 種の評価作業が行われ，TINMAN を満たす既存のプログラム言語は存在しないとの結論が下された．

1977 年になると，Revised IRONMAN を満足する新言語設計の作業が，4社競作の形で Cii-Honeywell-Bull, Intermetrics, Softech, S. R. I. International の各機関に依頼された．設計された新言語は，それぞれ順に Green 言語，Red 言語，Blue 言語，Yellow 言語と仮に呼ばれた．

1978 年3月の選択で，Blue 言語と Yellow 言語が

† Notes on Ada Implementability Issues by Takehiro TOKUDA (Department of Information Science, Tokyo Institute of Technology).

†† 東京工業大学理学部情報科学科

* Ada [éidə]

落選し，1979年5月の最終選択で Red 言語が落選した．結局，米国国防総省の共通高水準言語として，Green 言語が選ばれ，史上最初の女性プログラマといわれる Ada Augusta Lovelace にちなみ，Ada と命名され，現在に至った．

Ada は1980年初めに予定されている最終仕様の凍結まで，テストと評価の時期にあり，米国・欧州各地で，コンパイラの試作や評価作業が展開されている．Ada コンパイラの試作は，例えば米国の陸・海・空軍のそれぞれで，また Carnegie-Mellon 大学，New York 大学，Brown 大学等で，更に欧州では Karlsruhe 大学等で行われている．なお特に，Honeywell 社では属性文法を用いて記述した Ada コンパイラが Multics の下で動いており，処理速度は低速であるが，Ada の講習会や評価作業に利用されている．

Ada 支援環境の要求仕様を定義するプロジェクトも1978年から開始され，SANDMAN，PEBBLEMAN[6] として，発表されている．なお，1979年9月以降には，更新された PEBBLEMAN も発表される予定である．

また，Ada のセマンティックスを，denotational semantics を用いて形式的に記述しようとするプロジェクトも進行中である．

以上が Ada の現況であるが，今後の Ada 並びに PEBBLEMAN の関連プロジェクトについては，文献14) 及び 8),9) を参照されたい．なお他に追記すべき事項としては，日本国内でもいくつかの Ada 研究グループにより処理系の試作が進められている（例えば文献 3) など）．また米国国防総省は，Ada の最終仕様の凍結後，Ada コンパイラの作成と検定のプロジェクトが済み次第，比較的低廉な価格で，Ada コンパイラの配布を行い，Ada の普及を図る予定である．

3. オーバローディング

Ada 資料[7),12)] に従い，オーバローディングの内容を紹介し，次に Harald Ganzinger[10)] に従いその判定実現法を示し，最後に本報告者のコメントを添える．

3.1 オーバローディングの内容

同一の名前等が，共通の宣言有効範囲内に存在して，しかも2種類以上の異なる意味を表わすことができる性質をオーバローディングと言う．Ada では，手続き名，関数名，演算記号，リテラル，アグリゲート (Man 3.6.2)，エントリ名 (Man 9.5) がこの性質を持つ．プログラム中に，オーバロードされた名前等が出現している時は，その使用位置の文脈から意味が一意的に確定しなければならない．従ってプログラマは，オーバロードされた名前等の表わす意味が一意的となるよう，十分な文脈情報をプログラム中に盛り込む必要がある．文脈から一意的意味が確定しない時，そのプログラムはエラーとなる．

宣言例 1

```
type COLOR1 is (RED, YELLOW, GREEN);
type COLOR2 is (GREEN, BLUE, BROWN, YELLOW);
```

宣言例 2

```
function "+" (X: INTEGER) return INTEGER;
function "+" (X, Y: INTEGER) return INTEGER;
function "+" (X, Y: FLOAT) return FLOAT;
```

宣言例 3

```
procedure F (A: in COLOR1);
procedure F (A: out COLOR1);
procedure F (B: in COLOR2);
```

宣言例 4

```
type MY_INTEGER is new INTEGER;
C: COLOR1;
```

オーバローディングを伴う宣言のいくつかを宣言例1—宣言例4に示す．宣言例1では，列挙リテラル (Man 3.5.1) である GREEN と YELLOW が，共に COLOR 1 型と COLOR 2 型に属し，オーバロードされている．同様に，宣言例2では演算記号＋が，宣言例3では手続き名 F がオーバロードされている．

以上3例では，オーバローディングが伴っていることは一目瞭然であったが，そうでない宣言もある．宣言例4のように，プログラマが新しい型 MY_INTEGER を INTEGER 型からの導出型 (Man 3.4) として定義すると，数値リテラル 10 やアグリゲート (71, 92, 53, 14) にオーバロードが生じ，また INTEGER 型に対して定義されていた演算記号，副プログラム名にもオーバロードが生じるので，注意しなければならない．

次に，オーバロードされた名前等の使用例を見てみよう．宣言例1—宣言例4の環境の下で，次のような代入文を考えてみよう．

```
C := GREEN;
```

この場合，右辺の範囲内では，GREEN が COLOR1 型のリテラルか COLOR2 型のリテラルかは区別がつかず，あいまいである．しかしながら，左辺の変数 C が右辺に要求する型は COLOR1 型であるから，右辺の GREEN は COLOR1 型のリテラルであることが判明す

る．以上で一意的意味が確定したから，正しい使用例であるということになる．

同様に以下の例では，

```
F(BLUE);
```

宣言例3の3番目の手続き F に対し，COLOR2 型のリテラル BLUE を実引数として，手続き呼出しを行っていることが，一意的に確定する．

次の例では，仮引数と実引数の結合の方向 (Man 5.2.1, Man 6.3) により，手続き名 F がそれぞれ1番目の F，2番目の F と確定する．

```
F(A := C);
F(A := C);
```

また次の例では，仮引数の名前から手続き名 F は3番目の F であることがまず判明し，次に3番目の F の定義から，実引数の GREEN は COLOR2 型の GREEN であることが確定する．

```
F(B := GREEN);
```

また更に，次のような文脈における GREEN も，リテラル BROWN が COLOR2 型にのみ属している事実から，COLOR2 型の GREEN であることが確定する．

```
for J in GREEN..BROWN
```

もう少し複雑な例として，宣言例5の環境の下で次のような代入文を考えると，

宣言例 5

```
type T1 is
  record
    A: constant range 1..2;
    case A of
      when 1 => I: INTEGER;
      when 2 => B: BOOLEAN;
    end case;
  end record;
type T2 is
  record
    A: constant range 1..2;
    case A of
      when 1 => B: BOOLEAN;
      when 2 => I: INTEGER;
    end case;
  end record;
function G (Z:T1) return INTEGER;
function G (Z:T2) return INTEGER;
X: INTEGER;
```

```
X := G(1, X));
```

同様に関数名 G は1番目の G で，その実引数は T1 型のレコード・アグリゲート (Man 3.7.3) であることが確定する．

以上の例からわかるように，オーバロードされた名前等からその意味を判定するのに用いることのできる情報は，多様な形態を有している．

また言語仕様上，オーバローディング可能な名前等はかなり広い範囲であるが，変数名，定数名，タイプ名等にはオーバーローディングが許されていないことに注意されたい (Rat 7.5).

3.2 判定実現法

Ada において，オーバロードされた名前等を含んだプログラムの構成要素 (program construct) が，その名前等を正しく使用しているための条件を述べると，次のようになる．

"その構成要素のすべての部分構成要素において，型の関係が宣言されている関係に矛盾せず，しかも一通りの解釈が成立する．"

この条件を基礎として，以下でオーバロードされた名前等の判定実現法を示す．以下の方法は Ganzinger[10] らの方法を最も原理的な形で表現したものである．

[判定実現法]

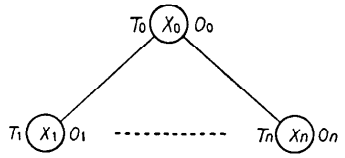
(1) 与えられた構成要素Cに対して，そのすべての部分構成要素の包含関係を示す解析木 C* を作る．解析木 C* のそれぞれの節は部分構成要素に対応している．

(2) 解析木 C* のすべての部分構成要素 X に対し，次の集合 O_X と T_X を対応させる．

O_X: X で表示できる操作をすべて集めた集合．

T_X: O_X 内の操作で得られる結果の型をすべて集めた集合．

(3) 解析木 C* 内のすべての下図のような親子関



係で，次の関係 R が成立するまで，それぞれの O_X, T_X から不必要な元を除去して行く．

$(t_0, \cdots, t_n) \in T_0 \times \cdots \times T_n$ が関係 R を満たす．

$\Rightarrow O_0$ の元 o が存在して，t_0 は o の結果の型，$t_1, \cdots, t_n$ はそれぞれ o の1番目，…，n 番目の引数の型と

宣言例 6

```
type T1 is
  record
    S: INTEGER;
    R: INTEGER;
  end record;
type T2 is array (1..2) of INTEGER;
function F (P: T1) return T1; --F1
function F (P: T2) return T2; --F2
X, Y: INTEGER;
A: T1;
```

なりうる.

(4) 解析木 C* のすべての部分構成要素 X について, O_X と T_X がただ1つの元から成っている時, 与えられた構成要素 C は一意的意味を持っていたことになる. それ以外の時, エラーを宣言する. □

宣言例6の環境の下で, 次のような代入を行った場合の判定例を図-1 に示す.

```
A := F(X, Y);
```

なお図-1 では, 最終的に残った操作とその結果の型に下線を施してある.

以上の例では, ステップ (3) の親子関係のチェックを上方から下方へ1パス行うだけで, 解釈が確定したが, 一般にはステップ (3) を何らかの順序で, T_X と O_X の値が収束するまで何回でも繰返さなければならない. しかしながら, Ganzinger によると評価の回数としては4パスで十分だそうである.

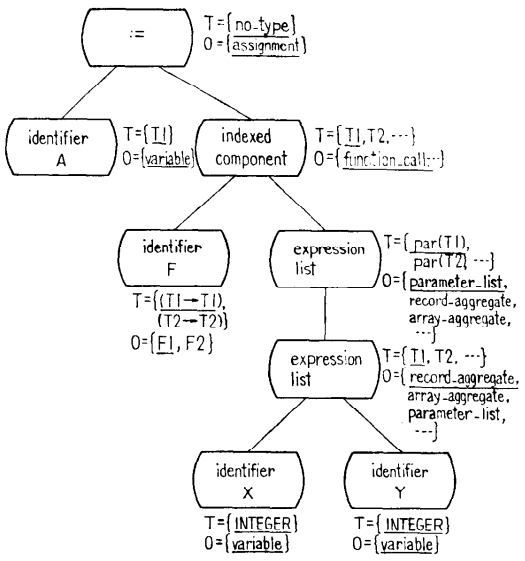


図-1 オーバローディングの判定例

3.3 オーバローディングに関するコメント

プログラミングにおいて, 内容を反映した良い名前を使う自由度を高めるため (Rat 7.5) に導入されたオーバローディングであるが, 結果的には読み易さの低下, 誤りを含むプログラムが他の解釈で偶然に実行されてしまう可能性の増加, 判定法実現によるコンパイラの増大化を招いていることは否定できない.

しかしながら, オーバローディングのデザイン選択が, 他のデザイン選択 (タイプ宣言ごとに新しいタイプが作られる. 複数のタイプ用や引数任意個の副プログラムは作れない.…) と密接な関連を持つため, Ada からオーバローディングを取り去ることは, 言語全体のデザインの大幅な変更を意味することとなる.

従ってオーバローディングに対する当面の対策としては, 節度あるオーバローディングの利用を訴え, 紛らわしいリテラルには型の積極的指定 (Man 4.6.1) を推奨する以外になさそうである.

4. ランデブ

ランデブの内容を簡単に紹介し, 次に Ada デザイン釈明書[12] で示唆している実現法の輪郭と, Olivier Roubine[15] の報告に従い, Habermann-Nassi の実現法を紹介する. 最後に本報告者のコメントを添える.

4.1 ランデブの内容

並行に実行されている2つのタスク間で起こる交渉を Ada では, ランデブ (Man 9.5) と呼ぶ. ランデブは, ランデブ要求側がエントリ呼出しを行い, ランデブ受付側がそのエントリ呼出しをアクセプトすることによって開始する. 従ってランデブが開始されるためには, 要求側タスクの制御がエントリ呼出しに達し, 受付側タスクの制御がそのアクセプト文に達していなければならず, いずれか一方が先に到達しても, 他方が到達するまでは, エントリ呼出し動作もアクセプト動作も完了しないで待ち状態となる.

ランデブが開始されると, 要求側タスクは待ち状態に入り, 受付側タスクは呼ばれたエントリの本体部分の実行を開始する. 受付側タスクがエントリ本体部分の実行を完了すると, ランデブは終了し, それぞれのタスクは独自に実行を再開する. すなわち要求側はエントリ呼出し文の次の位置, 受付側はアクセプト文の次の位置から実行を再開する.

このランデブを用いると, 従来の同期問題, 通信問題, 資源保護問題等を解決できるわけであるが, 以下

宣言例 7

```
task SEMA_R is
  entry P;
  entry V;
end SEMA_R;
task body SEMA_R is
begin
  loop
    accept P do
      null;
    end;
    accept V do
      null;
    end;
  end loop;
end SEMA_R;
```

では排他制御問題を例に用いて，ランデブの例を示そう．今，特定の資源Rを複数のユーザタスク間で，同時に使用しているユーザタスクの個数が常に高々1個であるように使いたいとする．この排他制御問題は，例えば資源Rに1つの2値セマフォア（値が0と1のみ）を付与しても解決するが，Ada では例えば次のようにして解決する．

資源Rの制御を担当するタスクモジュール SEMA_R を，宣言例7のように定義し，各ユーザタスクは，以下のように資源Rの使用直前にエントリPを呼出し，使用直後にエントリVを呼出せばよい．SEMA_R 本体の上を走る制御は，ただ1つであり，しかも順次的にしか走らないから，これで相互排反が保証される．

```
...
SEMA_R. P;
(資源 R の使用)
SEMA_R. V;
...
```

この例では，エントリPもエントリVも本体（do と end で囲まれた部分）が空となっているが，エントリ呼出しが引数を持つ場合，本体部分でそれらの引数に関する作業を行う．なお，do null; end は省略できる．

複数のユーザタスクが，エントリPの呼出しを行うと，高々1つの呼出ししかアクセプトされないから，いくつかのタスクは呼出し動作が未完了状態で，将来アクセプトされるまで，待たされることになる．これらのタスクは，エントリに付与されているキュー内で，呼出し順に並べられて待っており，キューの先頭のものから順に将来のランデブの機会を得る．

4.2 実 現 法

Ada デザイン釈明書[12] では，11.5 節の11ページを多重タスキング実現法のスケッチにあてている．本稿の目的の範囲内で，ランデブの実現法の部分をまとめると以下のようになる．

［エントリに必要な情報］

各エントリ E は次の3種の情報が必要である．

① エントリの状態

エントリ E を所有するタスクが，E のアクセプト文に達しているが，E に対するエントリ呼出しはまだない時，“開”状態．これ以外の時は，“閉”状態．

② キュー（Q_E）

エントリ E に対してエントリ呼出しを行ったタスクを，呼出し順に並べる待ち行列．

③ 転送先番地（A_E）

実行すべき E の本体の先頭番地．

［アクセプト文の実行］

タスク T がエントリ E のアクセプト文に到達したら，次の動作を行う．

（1） エントリ E のキュー Q_E が空であれば，(2)—(4) を，空でなければ（4）を実行する．

（2） エントリ E の状態を“開”とし，現在のプログラム・カウンタの値を転送先番地 A_E に格納する．タスク T を休止状態とする．

（3） タスク T が休止状態から活動状態に移行したならば，次のステップ（4）へ．

（4） Q_E の先頭位置のタスクを Q_E から取り去り，エントリ E の本体を実行する．

［エントリ呼出しの実行］

タスク S がタスク T のエントリ E を呼出したら，次の動作を行う．

（1） エントリ呼出しの実引数を評価し，然るべき領域に格納する．

（2） タスク S を休止状態とし，Q_E の最後の位置に S を並べる．

（3） エントリ E の状態が“閉”であれば，何もしない．状態が“開”であれば，タスク T を休止状態から活動状態へ移行させる．

［ランデブ完了］

（1） タスク T がエントリ E の本体の実行を終了したら，エントリ呼出しを行った側であるタスク S を，休止状態から活動状態へ移行させる．□

以上から明らかなように，ランデブに対する要求側と受付側の関係は“主従関係”でない．またデザイン意図的にも，手続き呼出しがいわば呼出し側の制御の“出張”であるのに対し，エントリ呼出しはあくまで“代行依頼”に過ぎず，エントリ本体の実行は受付側に“代行”されるという特徴を持つ．

しかしながら，以下で説明する Habermann-Nassi 法は，このエントリ呼出しの実現を，手続き呼出しに還元してしまう簡便な方法である．基本的アイデアは次の通りである．

エントリ呼出しを手続き呼出し，エントリ本体の実行を手続き本体の実行と置換えると，まず困るのが，要求側の一方的希望でランデブが実現してしまうことである．そこで，手続き本体と本体以外のコード各部分にロックを施し，ただちには実行できないようにする．次に，見かけ上受付側タスクの本体の上を1つの制御が順次的に移動するように，ロックの開閉を工夫する．

以下，実例で示そう．

① 最も単純な場合

図-2(a)のように，受付側タスクの本体が，アクセプト文の連続である場合は，**図-2**(b)のように，アクセプト文の個数＋1個だけ2値セマフォアを用意すればよい．

② 少し複雑な場合

図-3(a)のように，アクセプト文とアクセプト文の間に一般的文がはさまっている場合は，**図-3**(b)のように，それぞれの一般的文の最後から，次の一般的文の先頭へ go to 文で制御を移しておけばよい．

③ 少し一般的な場合

図-4(a)のように，セレクト文 (Man 9.7) のガードがすべて TRUE の場合は，**図-4**(b)のように，セレクト文全体に1つの2値セマフォアを用意し，呼出したエントリに応じてその本体の先頭へ分岐すればよい．

```
task body T is
begin
  loop
    accept E1 do
      ①
    end E1;
    accept E2 do
      ②
    end E2;
    accept E3 do
      ③
    end E3;
  end loop;
end T;
```

(a) 最も単純な場合

```
S1 := 1;
S2, S3 := 0;
S0 := 0; P(S0);
E1: P(S1);
    ①
    V(S2);
    return;
E2: P(S2);
    ②
    V(S3);
    return;
E3: P(S3);
    ③
    V(S1);
    return;
```

(b) 変換後の様子

図-2

```
task body T is
begin
  loop
    ①
    accept E2 do
      ②
    end E2;
    ③
    accept E4 do
      ④
    end E4;
  end loop;
end T;
```

(a) 少し複雑な場合

```
S1:=1;
S2, S3, S4:=0;
L1: P(S1);
    ①
    V(S2);
    go to L3;
E2: P(S2);
    ②
    V(S3);
    return;
L3: P(S3);
    ③
    V(S4);
    go to L1;
E4: P(S4);
    ④
    V(S1);
    return;
```

(b) 変換後の様子

図-3

```
task body T is
begin
  loop
    select
      when TRUE => accept E1 do
                     ①
                   end E1;
    or when TRUE => accept E2 do
                     ②
                   end E2;
    end select;
  end loop;
end T;
```

(a) 少し一般的な場合

```
S:=1;
S0:=0; P(S0);
E1: P(S);
    go to L1;
E2: P(S);
    go to L2;
L1: ①
    V(S);
    return;
L2: ②
    V(S);
    return;
```

(b) 変換後の様子

図-4

④ 一般的な場合

以上の考え方を進め，各エントリにロックを1つ，転送先アドレスを1つ持たせることにより，セレクト文が一般的ガードを持つ場合や同一エントリに対して2個以上のエントリ本体を持つ場合等の一般的場合も処理できる．

4.3 ランデブに関するコメント

初期 Green 言語におけるランデブは C. A. R. Hoare の Communicating Sequential Processes[11] の“悪しき導入”の感があったが，Ada におけるランデブはずっとすっきりしたものになったと思う．

しかしながら，実際にランデブを実現しようと思うと，本稿の議論の外に，セレクト文の実現，タスキングの例外処理等の“試練”が控えている．従ってこれらの総合的議論なしに，ランデブを評価することは控えねばならない．

5. むすび

Ada 実現に関する3つの講演[1),10),15)]から，Ada 現況，オーバローディング，ランデブの3つの話題を取り上げ，Ada 資料[7),12)]に基づいて再整理しながら，報告した．従って本稿中の歴史的事項や技術的事項に関する誤りは，本報告者の責任であることを付記しておきたい．本稿が Ada の評価をめぐる議論の参考になれば幸いである．

本稿の作成にあたり，日本国内のいくつかの Ada 研究グループの方々に御教示を頂いた．特に，米田信夫先生，斉藤信男先生，筧捷彦先生，原田賢一先生，吉田裕之君，井上謙蔵先生に感謝する．また，深夜から早朝にかけて OHP 原稿コピーの手配をして下さった Colorado 大学の Leon Osterweil 先生に感謝する．

参考文献

1) Brosgol, Ben: Ada Implementability Issues, 1979年8月9日に Colorado 州 Denver で行われた講演.
2) Buxton, John: Pebbleman, 1979年8月21日と22日に California 州 Santa Cruz で行われた講演.
3) 近山隆：プログラム言語 ADA 処理系の試作，第21回プログラミング・シンポジウム予稿集 (1980).
4) Dijkstra, E. W.: On the GREEN Language submitted to the DoD, SIGPLAN Notices, Vol. 13, No. 10 (Oct. 1978).
5) Department of Defense: DoD STEELMAN requirements for high order computer programming languages (June 1978).
6) Department of Defense: DoD PEBBLEMAN requirements for the programming environment for the common high order language (July 1978).
7) Department of Defense: Preliminary ADA reference manual, SIGPLAN Notices, Vol. 14, No. 6 (June 1979).
8) Department of Defense: Ada test and evaluation newsletter, Number 1, SIGPLAN Notices, Vol. 14, No. 9 (Sept. 1979).
9) Department of Defense: Ada newsletter, Number 2, SIGPLAN Notices, Vol. 14, No. 10 (Oct. 1979).
10) Ganzinger, Harald: Some aspects of operator identification in Ada, 1979年8月9日に Colorado 州 Denver で行われた講演.
11) Hoare, C. A. R.: Communicating sequential processes, Comm. ACM, Vol. 21, No. 8 (Aug. 1978).
12) Ichbiah, J. D., et al.: Rationale for the design of the ADA programming language, SIGPLAN Notices, Vol. 14, No. 6 (June 1979).
13) 上條史彦：プログラミング言語への期待—米国国防省の HOL プロジェクトについて—，情報処理，Vol. 19, No. 3 (1978).
14) 大野恂郎：Ada—米国国防総省の新言語—，bit, Vol. 11, No. 7 (1979).
15) Roubine, Olivier: (Non-exhaustive) list of issues in Ada implementation, 1979年8月9日 Colorado 州 Denver で行われた講演.
16) 安村通晃：米国国防省 (DoD) 規格による言語，情報処理，Vol. 20, No. 3 (1979).

付 録 Ada 実現上のいくつかの問題[1),15)]

(1) 語彙解析・構文解析上の問題
省略.

(2) 意味解析上の問題
タイプ同一性
導出型タイプへの属性の自動的相続
ジェネリック
副作用の検出
パッケージと名前の束縛
分離翻訳

(3) 表現選択・コード生成上の問題
ドープ・ベクトルの回避
式の評価順序
実行時のチェック
例外処理
記憶域管理
その他

付 記

本稿作成後，次の文献が出現した．なお文献 19) は，Prentice-Hall 社から1979年12月に出版される予定の本の第一章部分を掲載したものである．

17) 吉田裕之：コンパイラ自動生成系を用いたプログラミング言語 ADA 処理系の試作，東京工業大学情報科学科卒業論文 (1980年3月).
18) TSINKY：新しいプログラム言語 ADA，bit, Vol. 12, No. 2 (1980).
19) Wegner, Peter: Programming with ADA: An introduction by means of graduated examples, SIGPLAN Notices, Vol. 14, No. 12 (Dec. 1979).

(昭和54年11月26日受付)