20. プログラム言語ＡＤＡ処理系の試作

東京大学 工学部　近山　隆
Takashi Chikayama

プログラム言語ADA〔1,2〕は米国防総省のSteelman要求〔3〕を満たすために設計された言語である。ADAはPASCAL〔4〕の要素の多くを受けつぎながら、データ抽象化、モジュール化の機能を付加し、並列制御の機構も備えている。その成立の経緯から、今後のプログラム言語に与える影響は多大なものとなることは明白である。このADAの処理系を試作した経験から得た問題点について述べる。

1. ADAについて

ADAは型の概念を持つ汎用高水準言語で、PASCALに代表される静的な型の検査を特徴とする言語族の一員である。ADAの「型」は型（type）と副型（subtype）のふたつの概念からなる。副型はとり得る値や配列の添字の範囲などを拘束するもので、実行時に動的に決まる。これに対して型は完全に静的に決まる。偶然同じ構造を持つようになった型同士は、別々に宣言してあれば違う型である。意図的に同じ構造を持つ異なる型を作ることもできる。

副譜名、演算子、列挙型の定数名の多重定義（overloading）を許すのもADAの特徴のひとつである。a＋bという場合の"＋"がa,bが整数の時は整数、実数の時は実数の加算を意味するのが多重定義である。ADAでは任意の利用者定義の副譜名、演算子、定数名を多重定義できる。例えば複素数型を利用者が定義する場合、実数型と同じSINという名の関数を定義するのは自由である。多重定義された名前は文脈から識別しなければならない。この点は処理系作成上の問題点のひとつである。

データ抽象化支援機能もPASCALにない機能である。ADAでは包（package）を用いて関連の深い型、データ、副譜をまとめて宣言できる。名前の可視性（visibility）を制御する機能も備わっており、プログラム単位間の分離度向上に役立つ。さらに分割コンパイル、譜庫（library）機能も静的な検査機能を損なわずに用いることができるので、協同作業によるプログラム開発に便利である。

機能面では例外（exception）の統一的扱いが特徴である。零除算など機械で検査される例外も、配列の範囲外添字などコンパイラが検査コードを出すものも、利用者が定義して算譜中に明記した例外も、すべて同様に扱うことができる。例外は副譜や区画（block）ごとに定められた例外処理譜（exception handler）によって認識、処理される。

並列制御も大きな特徴であるが、今回は組み込まなかった。

図1にPASCAL、SIMULA 67〔5〕、CLU〔6〕との比較を掲げた。全体の設計の可否はともかく、最近高水準言語に要求されている機能の多くを持っていることがわかる。

2. 言語仕様の制限

今回の処理系試作の目的は試作自体にあり、本格的利用ではない。そこで、処理系作成上の問題点があまりないと考えられ、しかも扱いが煩雑で作業量の多いものは割愛した。また、並列制御は操作系との関連が大きく、作業量も多くなるので、興味はあったが見送ることにした。

以下に今回除外した主な機能を列挙する。

並列制御

分割コンパイル

実数型

	ADA	CLU	PASCAL	SIMULA67
Storage Disipline	stack	heap	stack	heap
Separate Compilation	O	O	X	O
Array	O	O	O	O
Record	O	O	O	*
Variant Record	O	O	O	*
Enumeration Type	O	X	O	X
Data Abstraction	package	cluster	X	class
Generic Types	**	O	X	O
Pass-by-Value	?	X	O	O
Prallel Control Structure	?	O	O	O
Exception Handling	O	O	X	X
Iterators	X	O	X	X
Scope Control: Procedures	O	O	X	O
Scope Control: Variables	O	X	X	O

* : 抽象化機能で代行できる.
** : generic package で実現できる.
? : ADAでは用いる機構の指定はない.

図1. 諸言語の比較
(ADA以外は[7]による)

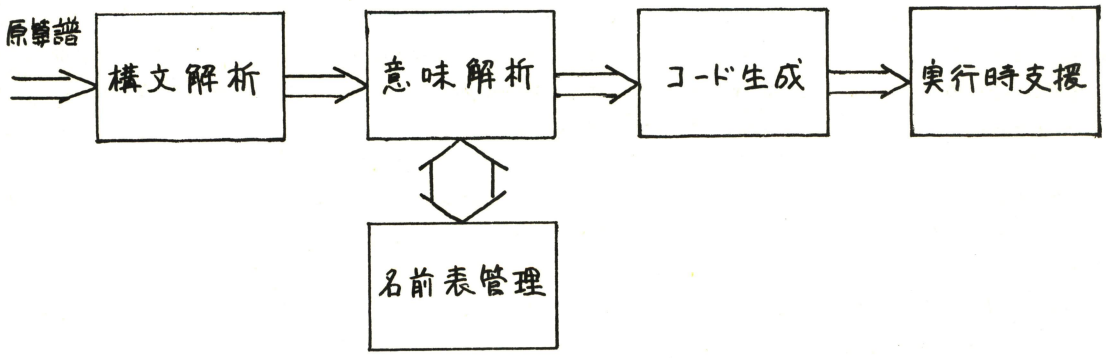


図2. 処理系の概要

```
X : array (1..15) of (RED, YELLOW, GREEN);
                  ↓
subtype G0000 is INTEGER range 1..15;
type G0001 is (RED, YELLOW, GREEN);
X : array (G0000) of G0001;
```

図3. 記述の標準化

表現指定 (representation specification)

I/Oの大部分

レコードの可変部

総称的宣言 (generic declaration)

3. 処理系の概略

処理系は図2のように五つの部分からなる。 各部の機能は次のようなものである。

1. 構文解析部　原始プログラムを読んで、 局所的な構文規則から判定できる範囲の解析を行ない、構文木を作る。
2. 意味解析部　構文木を受けとって、 多義名の認識や名前の属性に依存する構文解析を行ない、構文木を再構成する。
3. 名前表管理部　意味解析に必要な名前表を管理する。
4. コード生成部　完成された構文木から目的コードを生成する。
5. 実行時支援部　目的コードが動く環境を提供する。

この分割は分業を主目的として行なったのだが、 今後の改訂を容易にする意味もある。 例えば4、5を変更すれば多種の目的コードを生成することができる。

以下に各部分の問題点を概説する。

3.1. 構文解析部

この部分は従来のALGOL系言語と何ら変わるところはない。 原則として文脈独立な情報しか用いないので、例えば配列要素、 副譜呼び出し、 型指定式 (qualified expression) はいずれも〝name (expression)〟の形の構文を取り得るので、 区別できない。

構文上の便宜 (syntactic sugar) のいくつかはこの段階で発見され、 標準的な書き方に変換される。例えば変数宣言の中で暗黙に宣言された型は明白に宣言されたように扱う (図3)。 これは意味解析部の負担を軽減させるためである。

3.2. 意味解析部

意味解析部の主たる機能は名前とその表わす意味との対応づけである。 ADAでは多重定義が許されるので、名前と意味とは必ずしも一対一には対応せず文脈に依存する (図4)。 m重の多重定義が行なわれているn個の名前の意味が互いに依存している場合、 組み合わせの可能性はmのn乗になり、 単純なボトム・アップの算法ではコンパイル時間が問題になる。 そこで、 トップ・ダウンに情報を渡すことによって、 手間を少しでも省くようにしている。

3.3. 名前表管理部

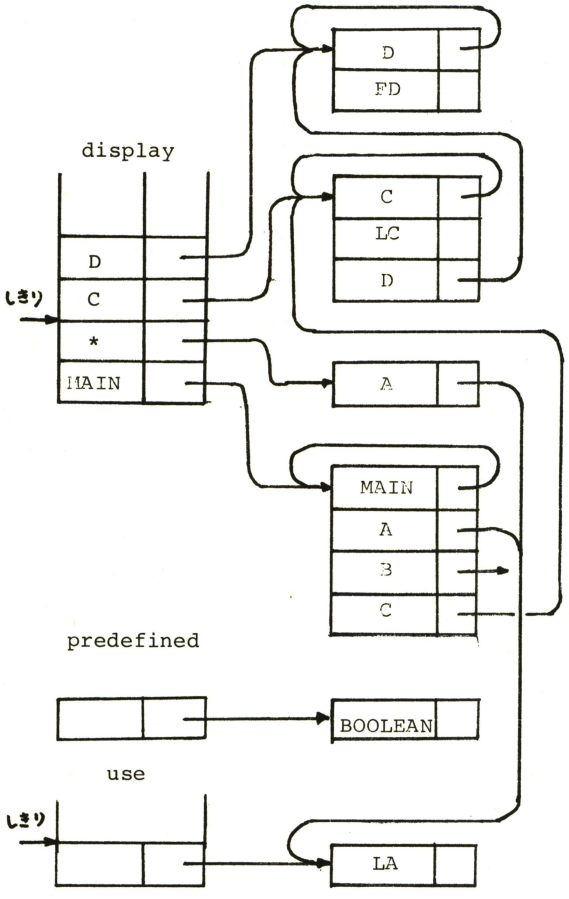
名前表は図5のような形をしている。 名前を探す手段は、 まずdisplay (通常の入れ子構造に対応) をしきりまで探し、 次に標準の環境、 最後にuse節のためのスタックをしきりまで探す。

可視性が制限される場合は、 このしきりの位置が移動し、 特に可視であると指定された単位名は別個に登録される。 use節はPASCALのwith文とにた機能を持つが、 with文と異なりそれまで見えていた名前を隠すことはないので、 最後に探すようになる。

```
function F (X : INTEGER) return BOOLEAN;    -- 1°
function F (Y : BOOLEAN) return BOOLEAN;    -- 2°
function F (Z : BOOLEAN) return INTEGER;    -- 3°

begin
  -- F(3)は 1°の Fで結果はBOOLEAN
  -- F(TRUE)は文脈によって 2°か 3° のどちらか
  -- F(F(3))は文脈によって 2°-1°または3°-1° の組合せ
  -- F(F(TRUE)) ?
  --    結果がINTEGERと決まれば3°-2° の組合せ
  --    結果がBOOLEANなら 2°-2°か 1°-3° があいまい
  -- F(F(Y:=TRUE)) とあれば2°-2° に決まる
end;
```

図4. 文脈に依存する名前




```
procedure MAIN is
  U : BOOLEAN;
  package A is
    LA : BOOLEAN;
  end A;
  package B is
    LB : BOOLEAN;
  end B;
  restricted (A)
  procedure C is
    use A;
    LC : BOOLEAN;
    restricted (C)
    procedure D (FD : BOOLEAN) is
    begin
      --
    end D;
  begin
  end C;
begin
end MAIN;
```

多重定義がある場合、 名前表管理部の返す値は可能性のある項目のリストになる。 多重定義がない場合は、use節で見えるようになった名前は互いに隠さないので、 同水準にあるかのように扱うことにして、 あいまいなものは誤りとする。

3．4． コード生成部

目的コードはHLISP[8]である。 今回の目標は目的プログラムの効率ではないので、 なるべく容易にコード生成ができるように考えたからである。 算譜中の名前にはLISPのアトムを対応づける。 文字列として同じでも意味の異なる名前には異なるアトムを対応させる。 このためLISPの動的結合則との矛盾は生じない。 また、ADAには副譜引数がないので、 環境切替 (context switching)がおきない。

変数や定数には成出 (elaboration)の際にリスト・セルを結合する。 機械語ならば記憶番地に対応するものである。 副譜名に対応するアトムにはその副譜の定義が属性として与えられる。 副型の制約 (constraint)や副譜の引数の既定値 (default value)にもアトムを対応させ、 成出の際に値を結合する。

例外と例外処理譜はCATCH-THROW制御[9]を用いた。 HLISPにはCATCH-THROWはないが、 ERRORSETを用いてこの機能を実現できる。

3．5． 実行時支援部

ADAでは標準の型、 演算子、 副譜と利用者定義のものとの間に区別はない (可視性だけは別)。 従って標準の演算子等も利用者定義のものとまったく同様に作るのが (効率を別にすれば) 良い。 それを定義するのがこの実行時支援部である。

入出力等の譜庫を用意するのもこの部分の役割であるが、 今回は最小限の入出力のみに限った。

4． 言語の問題点

試作の過程で指摘されたADA言語の問題点は多数ある。 その多くが手引書[1]が理由書[2]の言うようには〝complete〟でも〝concise〟でもない[10]ことからきた不備であるが、 好意的に行間を読み、 例題から推察しても、 なおかつ残った問題点も少なくない。 ここにそのいくつかをあげる。

4．1． 可視性

外側が見えないように制限された単位 (visibility list なしの restriction がついた unit) の中で、その単位自身の名前が見えることになっている。 この場合、 単位の外側の名前はいっさい見えないのだから単位名はそれ自身の内側になくてはならない。 理由書9-10頁の例を見ても、 単位名はそれ自身の〝local〟であることになっている。 もちろん、 一段上の単位の〝local〟でもある。 包Pの変数Xは P．X，P．P．X， P．P．P．X， ・・・のいずれを用いて表わしてもよい！？ このような規則を実現するのは煩雑である。

4．2． 成出の途中の例外

成出の途中で起きた例外は、 成出中の宣言並びに対応する例外処理譜が処理することになっている。 この例外処理譜からは成出に失敗したかもしれない変数名等が見えている。 成出の成否を調べる手段は理由書12．5．1に述べられているが、 はなはだ不自然であるし、 作譜者の責任とされている。 この問題は成出の

ひとつ外側の例外処理譜が処理にあたることにすれば自然に解決できる。　理由書には実現が困難であると記されているが、　特に困難はないと考えるので、　今回はそのように作った。

4．3．　多重定義

副譜の多重定義は強力で有用な概念統合の手段となり得る。　引数や結果の型によって行なわれる操作が変わるのは自然であろう。　しかし、　引数の渡し方（mode）や仮引数名の差だけで多重定義を行なうのは、　算譜をはなはだ読みにくくする危険がある（図4）。　副譜呼び出しに仮引数名や渡し方を書くことは、　引数の数が多くなる場合などには有用なのだが、　これは引数の対応づけのみに目的をしぼり、　引数や結果の型が同じ副譜名の多重定義は許さないのが良いと思われる。

5．　結　論

ADA言語には処理系作成の上での技術的困難は少ない。　言語は単一走査で解析しやすく設計されている。言語の仕様自体にも、　その記述にも、　数々の問題点が残っているが、　それらを解決すればかなり使いやすく強力な言語となり得るであろう。

この研究は東京大学大型計算機センターとの共同研究として行なった。　御協力いただいた大型計算機センターの諸氏、　処理系設計作成にあたった東大理学部情報科学科米田研究室の石畑　清、　池田　哲夫、　森本　真一市吉　伸行の諸氏、　筆者の研究室の和田　英一教授、　岩崎　博君、　その他御助言をいただいた多数のみなさんに感謝の意を表する。

REFERENCES

[1] Preliminary ADA Reference Manual, Sigplan Notices, Vol. 14, No. 6 (Jun. 1979), Part A

[2] Ichbiah, J.D., et al., Rationale for the Design of the ADA Programming Language, Sigplan Notices, Vol. 14, No. 6 (Jun. 1979), Part B

[3] Department of Defense STEELEMAN requirements for high order computer programming languages, Jun. 1978

[4] Wirth, N., The programming language Pascal, Acta Informatica, Vol. 1, No. 1 (1971), pp. 35-63, Springer Verlag

[5] Dahl, O.J., et al., The Simula 67 common base language, Pub s-22, Norweigian Computing Center, Oslo (1969)

[6] Liskov, B., et al., Abstraction mechanisms in CLU, Comm. ACM, Vol. 20, No. 8 (Aug. 1977), pp 564-576

[7] Hanson, S., et al., Summary of the Characteristics of Several "Modern" Progra-ming Languages, Sigplan Notices, Vol. 14, No. 6 (May 1979), pp. 28-45

[8] Kanada, Y., HLISP and supplementary HLISP-REDUCE manual, the Computing Centre of the Univ. of Tokyo (Feb. 1979)

[9] Moon, D., et al., "MACLISP Reference Manual", Project MAC, M.I.T. (1974)

[10] Dijkstra, E.W., On the GREEN Language submitted to the DoD, Sigplan Notices, Vol. 13, No. 10 (Oct. 1978), pp. 16-21