PASCAL シンボリック・デバッガの作成†

鈴木直也†† 萩原兼一††
荒木俊郎†† 都倉信樹††

高級言語を対象とするデバッガのあるべき姿の検討を PASCAL 言語仕様を例にして行い，得られた結果を指針として，小型計算機 NOVA 3 上で作成した PASCAL シンボリック・デバッガ (PSD) の外部仕様および実現方法を報告している．

高級言語用デバッガの問題点としては，a）ユーザにとって自然な実行の単位．b）スコープ規則に関する，変数や制御点の指定法，c）ユーザ定義型の値の表示法，d）動的記憶割付け領域上に構築される複雑なデータ構造の取扱い方および表示法，などを検討している．

PSD の特徴としては，1) CRT 端末を用いた対話型システムである，2) 豊富な指令が使用できる，3) すべての指令は，ソース・テキスト上の概念だけを用いて行えるので，ユーザは実行形式プログラムに関する情報を知る必要はない，4) ユーザ定義の型の変数値も，ソース・テキスト上の表現を用いて行う，5) 動的記憶割付け領域上の任意のデータ構造も，簡単な操作で表示できる，などがある．

1. まえがき

プログラムのデバッグを支援するシステムとして，デバッガが，特にアセンブリ言語を対象として広く用いられてきた．高級言語のプログラムに対してもデバッガを利用することは有用であるが，高級言語用デバッガには，アセンブリ言語用デバッガでは見られなかった問題点がある．

一般にデバッガの機能は次の2つに大別される．

i）プログラムの状態 (state) の表示・変更

ii）プログラムの実行の制御

アセンブリ言語用デバッガでは，これらの機能を実現するにあたり，メモリの番地や，メモリ・レジスタの内容を2進数やアセンブル時のシンボルを用いて操作できるようにすれば十分であろう．これは，アセンブリ言語のソース・プログラムと目的プログラムとの対応が単純で，プログラマがこれをよく理解しているからである．

これに対して，高級言語では一般にプログラマは，目的プログラムの形式を知らないので，上述の機能をソース・プログラム上の概念だけを用いて提供しなければ，真に有用なものとはならないであろう．ところで，高級言語においてはプログラムの状態が何から成っているかは自明ではない．したがって，デバッガの設計にあたっては，まずこの状態を定義することから始め，その定義がプログラマにとって自然なものであることが必要である．

たとえば，PASCAL プログラムの状態を定義するにあたっては次のような点が問題となる．

a）実行の単位とその指定　PASCAL は，入れ子構造をもち，FORTRAN などのように，単に文を実行の単位とすることはできない．

b）多様な形式の変数の取扱い　PASCAL には，整数型，文字型以外に，豊富なユーザ定義可能なデータ型がある．これらの変数値を，ソース・プログラム・レベルの表現で取り扱わなければならない．

c）名前の有効範囲規則 (scope rule) の取扱い　PASCAL の変数名にはそれぞれ有効範囲があるので，変数の指定は，単なる変数名だけでは曖昧である．

言語 PASCAL-∑C*[1),2)] を対象として，プログラマにとって自然な形で定義されたソース・プログラム・レベルの概念のみを用いて操作するシンボリック・デバッガ PSD[3),4)] (Pascal Symbolic Debugger) を作成した．シンボリック・デバッガとして十分な機能を果たすため，PSD は PASCAL のいくつかの特質（入れ子構造，豊富なデータ型，スコープ・ルール，ヒープ領域等）に対応している．また，ユーザが容易に必要な情報を収集できるような CRT 端末を用いた会話型システムとなっている．

† Development of PASCAL Symbolic Debugger by NAOYA SUZUKI, KENICHI HAGIHARA, TOSHIRO ARAKI and NOBUKI TOKURA (Department of Information and Computer Sciences, Faculty of Engineering Sciences, Osaka University).

†† 大阪大学基礎工学部情報工学科

* PASCAL-∑C と標準 PASCAL の相違点は次の2点である．
i）手続きごとの分離翻訳のため手続きの宣言部と定義部の分離．
ii）text 型以外の file 型，実数型をもたない．（詳細は文献 2) 参照）

高級言語を対象としたデバッガとしては，プロセス制御用言語 ESPRIT のための SOLDA[5]や，PASCAL プログラムの多彩な実行制御を特徴とした INSIDER[4] などがすでにあるが，PSD は上に述べたようにプログラムの状態をソース・プログラム上の概念だけを用いて簡単に扱えることを特徴としている．

2. PASCAL プログラムの状態

2.1 文 単 位

一般にプログラムの実行は，ある“最小単位”の実行による状態の遷移の連続と考えられる．しかし，**1.** で述べたように PASCAL ではこの単位が自明ではないので，次のように**文単位**と呼ぶ単位を定義した．

PASCAL プログラム・テキストの手続き（以下，特に区別の必要がないときは，PASCAL の procedure, function, program をまとめて手続きと呼ぶ）の実行部 (body) を，次に示す PASCAL の構文に基づいた規則で分割して得られる各部分を文単位と定義する．

ⅰ）代入文，手続き呼出し文，goto 文以外の文*を，**図1** で示した構文の Δ で示した部分で分割する．

ⅱ）分割した部分が文を含むときは，その文をⅰ）に従ってさらに分割する．

この文単位は，本システムでのプリティ・プリント形式で PASCAL プログラムを表示するときの各行に対応しているので，ユーザは正確な定義を知らなくても容易に操作できる．**図2** にその例を示す．

プログラム・テキストにおける文単位の識別は次の**文単位名**の形式で行う．

<文単位名>::=<所属手続き名>＋<文単位番号>

```
begin_Δ<s>{;_Δ<s>}_Δend
if <e>_Δthen <s>
if <e>_Δthen <s>_Δelse <s>
case <e> of_Δ<cl> : <s>{;_Δ<cl> : <s>}_Δend
while <e> do_Δ<s>
repeat <s>{;_Δ<s>}_Δuntil <e>
for <v>: = <e> to <e> do_Δ<s>
with <v> do_Δ<s>
```

<s> は statement, <e> は expression, <cl> は case label list, <v> は variable をそれぞれ表す．{ } は0回以上の繰り返しを表す．Δ が文単位の区切りである．

図 1 文単位の区切り方
Fig. 1 Definition of statement units.

* 手続きの実行部も1つの複合文 (compound statement) と見なす．

```
                procedure SCAN;
                  var X, I: integer;
SCAN +1         begin
SCAN +2           X: =0;
SCAN +3           for I: =0 to 10 do
SCAN +4           begin
SCAN +5             X: =DATA [I]+X;
SCAN +6             WRITELN (DATA[I])
SCAN +7           end
SCAN +8         end;
   ↑                      ↑
文単位名              ソース・プログラム
```

図 2 PSD が表示するリスティングの例
Fig. 2 Example of the listing displayed by PSD.

手続き* Q の実行部に含まれる n 番目の文単位を，文単位名 $Q+n$ で識別する．

2.2 実 行 点

プログラムの実行がどこで中断されているかを示すには，単に文単位名だけでは十分でなく，動的な手続き呼出しの過程を含めて示さなくてはならない．そのため**実行点**を，次の**文単位並び**の形式で表す．

<文単位並び>::=<文単位名>
|<文単位並び>/<文単位名>.

ただし，文単位並び $P_1+n_1/P_2+n_2/\cdots/P_i+n_i$ において，次の条件ⅰ），ⅱ）を満たさなければならない．

ⅰ）P_1 はプログラム名である．

ⅱ）$1 \leqq k \leqq i+1$ の各 k について，文単位 P_k+n_k が，手続き P_{k+1} の呼出しを含んでいる**．

実行点が，$P_1+n_1/P_2+n_2/\cdots/P_i+n_i$ であるとは，文単位 P_1+n_1 で手続き P_2 が呼ばれ，その中の文単位 P_2+n_2 でさらに手続き P_3 が呼ばれ … と手続き P_k まで呼出しが連鎖していることを示す．最後の P_k+n_k は，次に実行される文単位を示す．

2.3 変 数 指 定

PASCAL には，名前の有効範囲や手続きの再帰呼出しがあるので，単に変数名だけでは変数を一意に識別できない．ある変数を指定するには，その変数名の前に，その変数が利用可能になった時点の手続き呼出し関係をつけ加えた，次に定義する**変数指定**の形式で行う．

プログラム P のある実行点 $s=P_1+n_1/\cdots/P_i+n_i$ に対して，s における変数指定の集合 $VD_P(s)$ を次のような記号列の集りとする．

ⅰ）$1 \leqq k \leqq i$ なる k について，id が手続き P_k で宣言された変数名であるとき，記号列

* 手続き名はプログラム全体を通じて一意とする．
** 1つの文単位に同一の関数の呼出しが2つ以上含まれることはないとする．

$P_1/P_2/\cdots/P_k.id$ は $VD_P(s)$ に属する.

ii) i) で述べたもの以外を $VD_P(s)$ は含まない.

$VD_P(s)$ は，実行点 s で存在し得るすべての変数を表している．この中では，s のスコープでは見えないものも含まれている．すなわち，スコープ外の変数も状態の一部と見なす.

2.4 状　態

PASCAL プログラムの**状態** C は次のように，**実行点** s，**変数状態** V，**ヒープ関数** h の組からなる.

$C=(s, V, h)$.

変数状態 V は，次のような s における変数指定と，その値の組からなる.

$V=\{(v, \bar{v}) | v \in VD_P(s), \bar{v}=value(v)\}$

$value(v)$ は変数指定 v で指定される変数の値を示す.

ヒープ関数 h は動的記憶割付け領域（ヒープ）の状態を表すもので，nil 以外の不正でないポインタ型の値 $\bar{v}$ に対して，$h(\bar{v})$ はそのポインタの指す変数値を表す．(h の定義域，および $h(\bar{v})$ の値はプログラムの実行により変化する.)

以上に定義した $C=(s, V, h)$ で，PASCAL プログラムの実行状態がとらえられる．PSD の機能は，この状態の表示・変更，実行の制御をユーザが自然な形で行えるようにすることである.

3. PSD の外部仕様

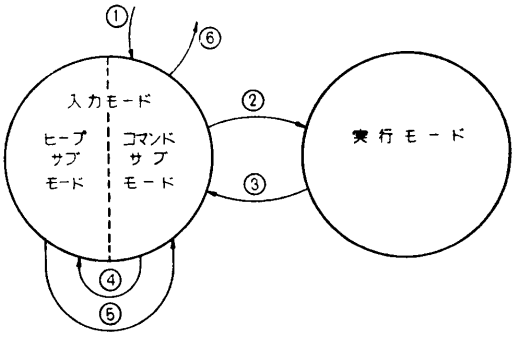
3.1 PSD のモード

PSD は CRT 端末を利用した会話型システムであり，ユーザの指示に従ってユーザのプログラム（以下 TP (target program) と略す）の実行の制御，状態の表示・変更を行う．ユーザから見て，PSD には次の2つのモードがある（**図3**）.

i) 入力モード　このモードでは，TP の実行が中断されていて，端末から PSD への指令を入力できる．状態の表示・変更の機能は主としてこのモードで提供される．入力モードはさらにコマンド・サブ・モード（**3.2 参照**）とヒープ・サブ・モード（**3.3 参照**）からなっている.

ii) 実行モード　このモードでは，TP が PSD の管理下で，実行される（**3.4 参照**）．TP の実行の制御という機能はこのモードで提供される.

3.2 コマンド・サブ・モード



遷移の原因

① PSD 起動	④ heat コマンド入力
② 実行の開始・再開	⑤ exit 入力
③ 実行の中断	⑥ PSD 終了

図 3 PSD のモード
Fig. 3 Mode of PSD.

表 1 PSD のコマンド
Table 1 Commands of PSD

構　文	機　能
reset	TP の初期化
run [to {<point>}]	TP の実行開始・再開，<point> の直前で実行中断
step [<num>]	1つまたは <num> 個の文単位を実行
return	手続きの途中から復帰
display {<disp-element>}	変数・ファイルの内容表示
state	実行点を文単位並びの形式で表示
assign {<v> : = <c>}	変数への定数代入
file {<file-id> : = <filename>}	PASCAL のファイルとディスク・ファイル，デバイスを結合
save <filename>	TP の状態をディスク・ファイルに退避
load <filename>	TP の状態をディスク・ファイルから復元
name {<identifier>}	名前の宣言情報を表示
log <filename>	PSD とユーザの会話をディスク・ファイルへ記録
source <point> to <point>	ソース・プログラムの表示
mark <markpoint> at <point>	文単位にラベル（マークポイント）をつける
list | activate | inactivate | clear [<num>]	ブランの表示，属性変更，消去を行う．(<num> はブラン番号)

[ ] 内は省略可，{ } 内は，で区切って繰り返し可，| は選択を示す.
<point> は文単位名か，<markpoint> [+<num>|−<num>] の形式
<num> は正整数，<v> は変数，<c> は定数を示す.

コマンド・サブ・モードは，ユーザと PSD の会話の中心となるサブ・モードであり，2種の指令（**コマンドとプラン**）が入力される．（プランは **3.4** で述べる．）

コマンドは入力されるとすぐ実行される指令で，状態の表示・変更，あるいは実行の開始・再開などを行う（**表1**参照）．TP の実行を指示するコマンドを入力すると実行モードに遷移するが，それ以外のコマンドではモードは遷移しない．

PSD のコマンドのうち特徴的なものを説明する．

・step　指定した個数の文単位を実行する．このとき文単位が手続き呼出しを含むときは，その呼出しから復帰までを含めて1つの文単位の実行と見なす．

・return　実行点が $P_1+n_1/\cdots/P_k+n_k$ のとき，最後に呼出された手続き P_k を呼出した文単位の次の文単位の直前まで実行する．

・display　変数の値を表示する．表示する変数の指定は前述の変数指定ではなく，その略記である次の**拡張変数名**を用いる．（{ }内は省略可．）

<拡張変数名>::=

{@<宣言手続名>{/<再帰カウント>}.}<変数名>.

実行点が $P_1+n_1/\cdots/P_k+n_k$ のとき，拡張変数名 $@P'/m.id$ は，変数指定 $P_1/P_2/\cdots/P_i.id$ を表す．

ただし，P_i は $P_1,\cdots,P_k$ の中で P' に等しいもののうち，右側から数えて m 番目のものである．

再帰カウントを省略したときは1と解釈する．また，宣言手続名を省略したときは，変数名はそのときの実行点のスコープに基づいて解釈される．

コマンド display では，拡張変数名にさらに↑や添字，フィールド名をつけて，ソース・プログラムと同じ形式で変数の指定を行うことができる（**図4**(a)）．

変数値の表示は，変数の型に応じて次の規則に従って表示する．

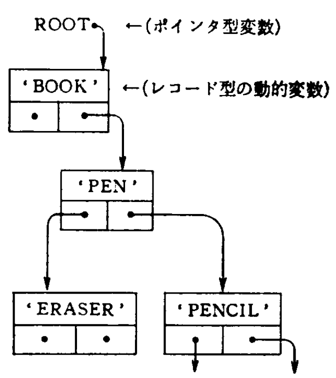
・スカラ型，集合型，文字配列型（これらの型およびポインタ型を，基本型と呼ぶ．）の変数については，ソース・テキスト中の各型の定数と同じ形式で値を表示する（図4(b)）．

・ポインタ型については，nil, not nil（動的変数を指している．）illegal（不正な値である．）のいずれであるかを表示する（図4(b)）．（動的変数を表示するには↑を用いた指定を行うか，ヒープ・サブ・モードで表示する．）

・配列型，レコード型の変数については，各要素あるいはフィールドごとに，添字やフィールド名とともに

```
>display @FACT/2.N
         ↑    ↑ ↑変数名
  宣言手続名┘    └再帰カウント
>display PTR↑.DATA[1]
(a) 様々な形式の変数指定の例
>display COLOR, COLSET, PTR
 COLOR=GREEN           (数え上げ型)
 COLSET=[RED, BLUE]    (集合型)
 PTR=not nil           (ポインタ型)
(b) 各タイプの値の表示の例
>display PERSON, DATA
 PERSON. NAME='SMITH'  ┐
       . AGE=25        │フィールドごとの値を表示
       . NEXT=nil      ┘
 DATA[1]=2             ┐
 DATA[2]=5             │要素ごとの値を表示
 DATA[3]=4             ┘
   ⋮
(c) レコード型，配列型の表示の例
 下線はユーザ入力を示す．
```

図 4 display コマンドの例
Fig. 4 Example of the display command.



(a) ヒープに作られるデータ構造の例

```
: from ROOT
      . WORD='BOOK'
#1    . LEFT=nil
#2    . RIGHT=not nil
↑―― (ポインタ番号)
#2 ←(次にたどるポインタ番号)
      . WORD='PEN'
#1    . LEFT=not nil
#2    . RIGHT=not nil
```

(b) ヒープ・サブ・モードで a) の構造をたどった例
(下線はユーザ入力を示す)

図 5 ヒープ・サブ・モードの例
Fig. 5 Example of the heap sub-mode.

に値を表示する．要素またはフィールドが基本型でないときは，さらに基本型まで分解して表示する（図4(c)）．

・テキスト型の変数については，その内容と読み書き位置(window)を表示する．また，読み書き位置を含む行のみを表示することもできる．

3.3 ヒープ・サブ・モード

動的変数については，コマンド display でポインタ変数を通して値を表示することもできるが，リストや木などの大きなデータ構造を表示するには指定が複雑になり不便である．ヒープ・サブ・モードでは任意のポインタ変数から始めてポインタをたどりながらデータ構造を表示することができる．**図5**はその例である．(a)のような木をたどって表示したのが(b)である．最初にたどり始めるポインタ変数を指定し，それ以後は，表示されるレコードのポインタ・フィールドに PSD により付けられるポインタ番号を入力して，そのポインタの指しているレコードを表示する．ポインタ番号の代りに0を入力すると，1つ前に表示した変数を表示する．また，現在のヒープ・サブ・モードになってからすでに表示した変数を再度表示するときは，何番目に表示した変数かユーザに知らせる．このようなバックトラックの機構によりどのようなデータ構造もたどることができる．

3.4 実行モード

実行モードでは，TP が PSD の管理化で実行される．PSD は次の機能を提供する．

i）プランの処理　**プラン**はコマンドと同じく入力モードで入力されるが，入力モードでは記憶されるだけで，実行モードで指定の条件が満たされれば実行される．プランは次の形式である．

<プラン>::=<条件部><命令部>

<命令部>::=<Pコマンド>|<命令部>；<Pコマンド>

TP の実行中，条件部（**表2**）で指定された1個，または複数の文単位が実行される直前に，命令部のPコマンド（**表3**）が PSD により実行される．

Pコマンドには次のようなものがある．

・**break**　TP の実行を中断して，コマンド・サブ・モードに遷移する．その際，実行点を文単位並びの形式で表示する．

・**trace, walk**　実行点 $P_1+n_1/\cdots/P_i+n_i$ に対して，文単位 P_i+n_i の部分のソース・プログラムを次次に表示する．また，walk ではさらに指定の秒数だけ，実行を一時停止する．

表2 プランの条件部
Table 2 Condition parts of the plan.

構　文	指定する文単位
at <point>	<point> で示される文単位
from <point 1>	<point 1> から <point 2> の間の文単位
to <point 2>	(2つ以上の手続きにまたがらない.)
in <procfunc>	手続き <procfunc> 中のすべての文単位
always	すべての文単位

表3 プランで用いられるPコマンド
Table 3 P-commands used in the plan.

構　文	機　能
break	TP の実行を中断
trace	次に実行する文単位を表示.
walk <num>	次に実行する文単位を表示して <num> 秒間実行を中断
check <v><op><c>	関係式が成りたてば何もせず，成りたたなければ実行を中断する.

<op> は関係演算子
上記以外に display, assign, save, return が P コマンドとして用いられる.

・check　ユーザの指定する変数と定数の簡単な条件式を評価して偽となれば，break と同じ処理をする．(たとえば at TEST+2 check X=1 など)

上記のほか，コマンド display, assign, save, return をPコマンドとして用いることができる．また，次のように，いくつかのPコマンドを組み合せて用いることができる．

at FACT+1 display N; assign FACT:=1; return

この例では，関数 FACT の最初の文単位で，パラメータを表示し，関数値を設定して，復帰する，といういわゆる“スタブ”の働きをする．

プランはコマンド・サブ・モードで入力されると，プラン番号が付けられてプラン表に登録される．プランは A (active) か I (inactive) のいずれかの属性をもち，I属性のものは記憶されているだけで実行の対象とはならない．プランの表示・消去はそれぞれコマンド list, clear で行い，属性の変更はコマンド activate, inactivate で行う．

ii）実行時エラー，強制中断の処理　オーバ・フロー等の PASCAL の実行時エラーが生じたときや，端末から CTRL/C が入力されて実行が強制的に中断されたときには，そのときの実行点を文単位並びの形式で表示して，会話モードに遷移する．ただし，実行時エラーが発生したときはその後，display などは可能だが，その実行点から TP の実行を再開すること

はできない.

4. PSD の実現

4.1 実現方式の概略

PSD は小型計算機 NOVA 3 (主記憶 96K 語, ディスク装置 (容量 20M バイト), CRT 端末装置を接続) 上, オペレーティング・システム MRDOS のもとで実現されている. 対象とする言語は PASCAL-∑C[1),2)] である.

PSD は, TP の実行の制御など目的プログラムに密着した処理を行う核部 (アセンブリ言語で記述) と, シンボル表現と内部表現の相互変換や, ユーザとの会話的処理を行う対話部 (PASCAL で記述, ただし MRDOS の提供する通信機能を利用する部分はアセンブリ言語で記述) の2つの部分からなっている. 核部と対話部の目的プログラムの大きさは, それぞれ2K 語, および 15K 語である.

PSD を用いてデバッグするためには, まずソース・プログラムを**図6**に示す手順でコンパイルする. PASCAL-∑C コンパイラは TP のソース・プログラムをコンパイルして, 中間言語であるPコード[7)] のプログラムを得る. このとき, 各文単位の間に対応する箇所に特別の Pコードを挿入する. このコードにより, 核部が TP の実行を制御する. (**4.2** 参照)

また, コンパイラは, 対話部が利用するため次の情報をディスク・ファイルとして出力する.

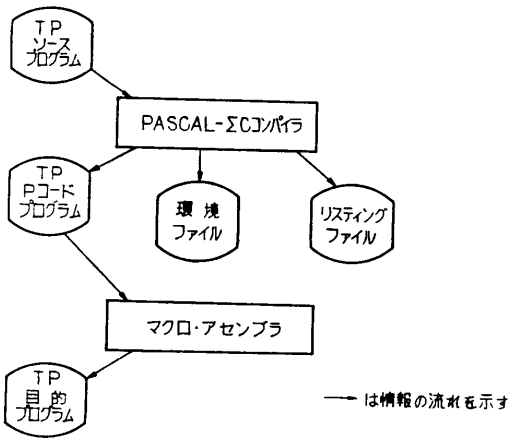


図 6 TP のコンパイル手順
Fig. 6 Compilation of user program.

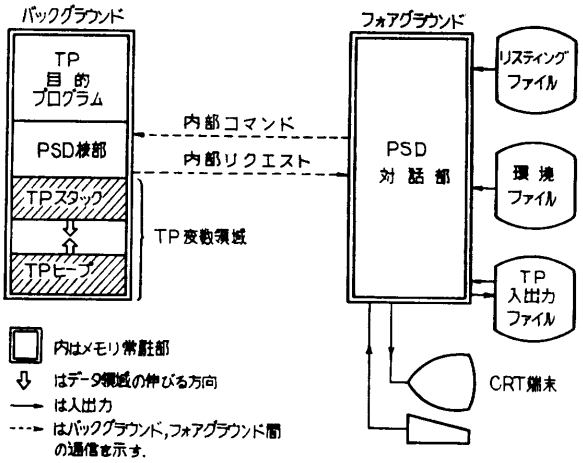


図 7 PSD によるデバック時の構成
Fig. 7 Configuration of PSD at debugging time.

i) 環境ファイル* TP に現われるすべての宣言の内容の表.

ii) リスティング・ファイル TP のソース・プログラムを1文単位が1行になるようにプリティ・プリント形式にしたもの.

TP の目的プログラムはPコード・プログラムをマクロ・アセンブラにより NOVA 3 の機械語に展開して得る. このとき, 前述の文単位の間に挿入された特別のPコードは核部を呼び出す命令に展開される**.

次に, PSD により TP をデバッグする際の処理方式について述べる.

オペレーティング・システム MRDOS では, 2つの異なるアドレス空間(バックグラウンド, フォアグラウンド) で, それぞれ1つのプログラムを並行して実行し, 通信領域を通じて通信することができる. PSD によるデバッグを行うときは, バックグラウンドに TP の目的プログラムおよび核部を結合したものをロードし, フォアグラウンドに対話部をロードして, 核部と対話部が通信しながら実行される(**図7**). TP と対話部はともに PASCAL プログラムであるが, NOVA 3 ではこれらを1つのアドレス空間 (32K 語) に格納するのが困難なので, この方式を採用した. また, TP と対話部が異なるアドレス空間で実行されるため, TP の暴走による誤動作の可能性は低い.

* 従来より手続きごとの分離翻訳のため出力されている[1),2)].
** デバッグ終了時には, このPコードを展開しないようにアセンブルすれば, TP 単独で実行される TP の目的プログラムが得られる.

PSD には5つの内部状態 P_I, P_k, E_k, E_I, E_T がある（**図 8**）．P_I, P_k は外部仕様の入力モードにほぼ対応し，E_I, E_k は実行モードに対応する．また，P_I, E_I では対話部が実行され，P_k, E_k では核部が実行され，E_T では TP が実行される．

E_k, E_T 間を除き，内部状態の遷移は内部コマンド（**表 5**）あるいは内部リクエスト（**表 4**）と呼ばれる対話部と会話部の間の通信を行うことで起こる．内部コマンドは，実行の再開を指示したり，TP の状態について情報を得たりするための，対話部から核部への要求である．内部リクエストは，実行モードでプランをサービスすべき条件が満たされた場合，実行時エラーが起こった場合，TP の入出力命令が実行された場合に，そのことを核部から対話部へ伝達する通信である．

4.2 実行モードの処理

実行モードでは主に核部が TP の実行を制御する．

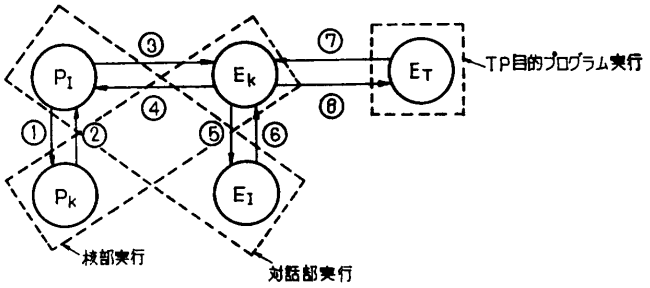
表 4 内部リクエスト
Table 4 Internal request of PSD.

	内部リクエスト名	機能
①	IERR	実行時エラーの発生を知らす．
	ISTEPR	内部コマンド ISTEP による1文単位の実行終了を知らす．
	IPLAN	実行すべきプランがあることをしめす．
②	IIN	ファイルから1文字入力する．
	IOUT	ファイルへ1文字出力する．
	IRESET IREWRITE	ファイルを reset, rewrite する．

① $E_K \rightarrow P_I$ の内部状態遷移をする．
② $E_K \rightarrow E_P$ の内部状態遷移をする．

表 5 内部コマンド
Table 5 Internal commands of PSD.

内部コマンド名	対応するコマンド	機能
IINIT	reset	TP の初期化
IRUN	run	TP の実行を再開する．
ISTEP	step	文単位をひとつだけ実行する．
IRETURN	return	手続きの呼出し点へ復帰する．
IADDRESS	display assign	スタック上の変数の絶対番地を知る．
IPEEK	display assign	指定した番地の内容を知る．
IPOKE	assign	指定した番地の内容を変更する．
ISAVE	save	TP の状態をディスク・ファイルに退避する．
ILOAD	load	TP の状態をディスク・ファイルから復元する．
ISTACK	state	現在の手続き呼出し関係を知る．



① TP を実行しない内部コマンド要求
② 〃 〃 応答
③ TP を実行する内部コマンド (IRUN, ISTEP, IRETURN) 要求
④ IPLAN, IERR, ISTEPR 内部リクエスト要求
⑤ 入出力内部リクエスト要求
⑥ 〃 〃 応答
⑦ 1文単位実行
⑧ TP 実行再開

図 8 PSD の内部状態とその遷移
Fig. 8 Internal states of PSD

i）実行点の把握　実行点 $P_1+n_1/\cdots/P_{i-1}+n_{i-1}/P_i+n_i$ を保持するために，部核はスタックとレジスタを持っている．$P_1+n_1,\cdots P_{i-1}+n_{i-1}$ の部分をスタックに，P_i+n_i をレジスタに格納している．前述の1文単位ごとに挿入されている命令がこれらの値を更新する．

ii）プランの処理　上記のスタック，レジスタが更新されると E_T, から E_k に状態遷移し，核部が実行すべきプランがないか調べる．実行すべきプランがなければ，E_T 状態に復帰する．実行すべきプランがあれば，内部リクエスト IPLAN が発せられ，P_I 状態に遷移し対話部に制御が移る．対話部はプランの命令部を後述（**4.3**）のコマンドと同様に処理し，その後必要なら内部コマンドにより TP の実行を再開する．

iii）入出力命令の処理　TP の入出力を PSD の端末に対して行えるように，TP の入出力を対話部が代行する．このため，TP の入出力命令が実行されると，それに対応した内部リクエストが発せられ，$E_T \rightarrow E_k \rightarrow E_I$ と状態遷移する．対話部が入出力の処理を終えると $E_I \rightarrow E_k \rightarrow E_T$ と状態遷移して TP の実行が再開される．

iv）実行時エラー処理　TP 実行中に PASCAL の実行時エラーが起こると，核部から内部リクエスト IERR が発せられ，エラーの発生箇所とエラー内容を対話部に知らせ $E_T \rightarrow E_k \rightarrow P_I$ と状態遷移する．対話部はエラー発生箇所とエラー・メッセージを表示する．

4.3 入力モードの処理

入力モードでは PSD は P_I, P_k 状態にあり，ユー

ザの入力するほとんどのコマンドは，対話部によりいくつかの内部コマンド（表 5）に変換されて核部へ送られる．核部からの応答は，対話部によりソース・プログラムのシンボルによる表現に変換されて表示される．

対話部は，ソース・プログラム・レベルの表現と内部表現を相互に変換するために，主記憶に読み込まれた環境ファイル（**4.1** 参照）を参照する．また，ソース・プログラムを表示する場合はリスティング・ファイル（4.1 参照）を利用する．

たとえばコマンド display で変数値を表示するときは，内部コマンド IADDRESS, IPEEK で変数の番地，値を順に知り，対話部がこれをソース・シンボルに変換して表示する．このとき，環境ファイルを参照する．

コマンド run, step, return は，それぞれ内部コマンド IRUN, ISTEP, IRETURN により処理される．このとき，$P_I \rightarrow E_k \rightarrow E_T$ と状態遷移し TP の実行が再開される．コマンド reset, display, assign, save, load も内部コマンドにより処理されるが，このときは $P_I \rightarrow P_k \rightarrow P_I$ と状態遷移し，核部による内部コマンドの処理が終わると対話部に制御が戻る．

5. あとがき

PSD は当初の目標である高級言語のシンボリック・デバッガとしての機能を実現していると思われる．PSD の利用により，プログラマはソース・プログラム上に定義された概念だけを用いてデバッグを行うことができる．PSD で用いた技法は，他の高級言語のシンボリック・デバッガにも応用可能である．

PSD の実現にあたって，PASCAL-∑C コンパイラの変更は，文単位ごとの特別のPコードの挿入，及びリスティング・ファイルの出力に関する部分のみであった．また，PSD 対話部を PASCAL で記述できたことは，開発費用の低減に役だった．

今後は，PSD, PASCAL-∑C 分離コンパイル・システム[8]，PASCAL の構文エディタ[9]を有機的に組み合せた PASCAL のプログラミング・システムを構成することを計画している．

謝辞 本システムの設計段階において多くの適切なご助言をいただいた鍵政豊彦氏（現在日立製作所）に感謝いたします．

なお，本研究は一部，文部省科学研究費補助金奨励研究（A）575235 の助成を受けている．

参考文献

1) 鍵政，荒木，都倉：手続きの分離コンパイル可能な PASCAL コンパイラ，情報処理学会第 20 回全国大会，4K-2 (1979).
2) 鍵政，荒木，都倉：PASCAL 内部手続きの分離翻訳，電子通信学会論文誌（D），Vol. J 63-D, No. 2, pp. 177-182 (1980).
3) 鈴木，鍵政，萩原，荒木，都倉：PASCAL のシンボリック・デバッガの作成について，情報処理学会第 21 回会国大会，2B-5 (1980).
4) 鈴木，萩原，荒木，都倉：PASCAL シンボリック・デバッガの作成，信学技報（オートマトンと言語），AL 80-40 (1980).
5) 春原，大井，関本，中村：高位言語デバッギング・システム SOLDA，情報処理学会論文誌，Vol. 20, No. 5, pp. 405-411 (1979).
6) 宮本，堀川：Pascal Machine を仮想したデバッグ・システムINSIDER，情報処理学会第 12 回記号処理研究会資料 80-12 (1980).
7) Nori, A. V., Ammann, U., Jensen, K., Nägeli, H. H. and Jacobi, Ch: The PASCAL ＜P＞ Compiler Implementation Notes, Revised Edition, Institute für Informatik ETH, Zürich (1976).
8) 菅井，岡根，荒木，都倉：対話型修正編集機能をもつ PASCAL 構文解析システム，情報処理学会論文誌，Vol. 22, No. 2 (1981).
9) 鍵政，萩原，荒木，都倉：PASCAL-∑C 分離コンパイルシステム，情報処理学会第21回全国大会，1B-7 (1980).

（昭和 56 年 2 月 20 日受付）
（昭和 56 年 5 月 20 日採録）