虫取り支援を目的とした PASCAL プログラム実行制御システム†

宮本衛市†† 堀川博史††† 高橋幸伸††

本論文はプログラムの虫取りを強力に支援し，さらにはプログラムの理解や診断を支援するためのツールとして開発した支援システム INSIDER について述べたものである．INSIDER は TSS 環境下で利用者に，いわゆる PASCAL マシーンとしての機能をソフトウェア上で模擬して提供する．すなわち，INSIDER のもとで，利用者はコンパイラの存在を一切意識することなく，PASCAL プログラムの実行制御等をソースレベルで指示することができる．INSIDER は文の並置構造と，手続きまたは関数の呼び出しを含めた動的な入れ子構造による構造的な実行・復元機能のほか，中断条件，打ち切り時間，異常事態の発生，あるいは端末からの割込みによる停止等，マシーンとしての多彩な機能も備えている．一方，変数の挙動の把握のため，指定された変数の値を実行時に逐次表示するほか，変数の静的およびポインタ変数による動的なデータ構造に応じた表示機能を有する．

INSIDER は PASCAL プログラムを翻訳して得られた機械語モジュール群をネットワーク構造のもとで掌握し，それを自身の制御下で実行させる．そこで利用者は，INSIDER にプログラムの実行・復元を指示し，プログラムの挙動を即座に観察することができるので，先入観や見落し等を排除し，強力にプログラムの診断あるいは誤りの原因究明等を遂行することができる．

1. はじめに

プログラムの作成に当っては，さまざまな誤りの混入を避けられず，誤りを発見・除去するためのテストあるいは虫取りが，プログラム作成に要するコストの中で大きな比重を占めている．もちろん，誤りの混入を引き起こさないプログラミング言語あるいはプログラミングの方法論の開発は必須の要請であり，これまでにも数多くの言語や方法論が開発あるいは提案されてきた[1]．しかし，プログラミングをするのが人間である以上，程度の差はあれ，依然として虫取りあるいはテストの過程は残らざるをえない．そこで，プログラマの主体性を認めた上で，計算機側でプログラマの活動を支援するための各種のソフトウェア・ツールの開発，さらにはそのシステム化が強く望まれているが[2),3]，いわゆるプログラミング・システムとしての条件を備えているのは，LISP, SNOBOL および APL に限られるとも言われているような現状である[4]．

本論文は，虫取りを強力に支援することを目的として開発した支援システムについて述べたものである．そもそも，コンパイラは一種のソフトウェア・ツールと考えることもでき，単に言語翻訳を行うばかりでなく，プログラム診断機能，虫取り機能，実行解析機能等を備えてシステム化したものが開発されている[5)~7]．しかし，虫取り機能を行使するために，あらかじめプログラム中に虫取りのための文を埋め込んでおくのでは即応性および融通性に欠け，またそのような文の挿入もわずらわしい．プログラミングと虫取りは判然と分けるべきであって，虫取りは，対象となるプログラムとは独立したソフトウェア・ツールで行えることが望ましい．また，そのようなツールとの間では，TSS 環境下で会話的に臨機応変の対応ができることが不可欠であるし，当然プログラムのソースレベルでの対応ができなければならない．このような虫取り支援を行うシステムとして，SOLDA[8]，AIDS[9]，PASCAL-I[10] などが報告されている．これらはいずれも TSS 環境下で，プログラムの実行制御，入出力動作，診断解析等をソースレベルで行うものである．

一方，本論文で述べる支援システム INSIDER (INteractive Structured Intelligent DEbugging Reinforced system) は，PASCAL プログラムを会話的，かつ構造的に実行制御し，虫取り支援，さらにはプログラムの理解や診断等を支援するソフトウェア・ツールである．INSIDER は TSS 環境下で利用者に，いわゆる PASCAL マシーンとしての機能をソフトウェア上で模擬して提供する．すなわち，利用者は PAS-

---

† Execution Control System for Aid of Debugging PASCAL Programs by EIICHI MIYAMOTO (Division of Information Engineering, Graduate School of Information Engineering, Hokkaido University), HIROSHI HORIKAWA (Information Systems and Electronics Development Laboratory, Mitsubishi Electric Corp.) and YUKINOBU TAKAHASHI (Division of Information Engineering, Graduate School of Information Engineering, Hokkaido University).

†† 北海道大学工学研究科情報工学専攻
††† 三菱電機(株)情報電子研究所

CAL のソースイメージのままでプログラムの実行・表示等を INSIDER に指示することができる．したがって利用者から見ると，端末をコンソールとした，一種の High-Level Language Computer[11] を専用に使うことのできる環境が与えられていると考えることができる．

虫取り支援機能としての眼目を，

（1） プログラムの実行をいかに制御し，場合によっては復元し，そして再開することができるか，

（2） 虫取りに有効な情報をいかに取り出せるか

の2点に集約して考える．(1)に関しては，INSIDER は PASCAL プログラムの構造に合わせた簡潔な実行・復元機能を備えており，中断時には任意の文を与えて実行することもでき，変数の監視・変更等を行うことができる．一方，(2)に関しては，指定した変数の値を実行時に逐次表示するほか，中断時には任意の変数の静的あるいはポインタ変数による動的なデータ構造に応じた表示が可能である．

INSIDER は PASCAL コンパイラを内蔵しており，翻訳して得られた機械語モジュール群をプログラムの構造に応じたネットワーク構造に展開して掌握している．したがって，INSIDER は端末からの指示に従い，機械語モジュール群を自身の制御下で実行させることができる．また，INSIDER はプログラム内の識別子等の翻訳情報を，木構造を基本とするデータ構造にすべて保存しており，PASCAL の文，式あるいは変数をオペランドとしてもつコマンドに対しても，プログラムの任意の箇所を想定して翻訳し，対処することができる．

以下，2章では INSIDER の諸機能について，3章では INSIDER の内部処理方式について述べ，4章では INSIDER の使用例を紹介する．

2. INSIDER の支援機能

2.1 プログラム構造に基づく制御機構の設定

INSIDER のもとでは，PASCAL プログラムのブロックの実行本体に対し，文の並置構造と，文の再帰的定義による入れ子構造を基本的な制御機構として考える．さらに後者に対しては，手続き呼び出し文で呼び出された手続き，あるいは関数呼び出しで呼び出された関数の実行本体が動的に入れ子構造に組み込まれていくものと考えて，この制御機構をプログラム全体の実行に拡張する．このような2重構造の制御機構でとらえたプログラムを実行制御するための単位として，文と節を考える．

文は並置構造に対応させるもので，PASCAL の構文規則による文である．そこで，ある文の実行というときには，その文が入れ子に含むすべての文，さらにそこで手続きまたは関数を呼び出していれば，それらすべての実行を意味する*．

一方，節は手続き呼び出しを含めた入れ子構造の実行を制御するために導入したものである．いま，文 A が文 $B_1, \cdots, B_n$ ($n \geq 0$，ただし $n=0$ のときは入れ子の文を含まないものとする）を入れ子に含むとき，A を $B_1, \cdots, B_n$ と，それらを A の種別に応じて実行制御する部分，すなわち A の枠組とに分けて考える．この枠組のもとで $B_1, \cdots, B_n$ を実行させることにより A の実行が行われ，一方 $B_1, \cdots, B_n$ は枠組の存在を意識することなく，それぞれの文を実行することができる．このような枠組を1個以上の節を設定することにより実現する．節は1つの実行単位で，基本的には式の実行に対応する．この節の設定を，入れ子に含まれている文に再帰的に適用することにより，結局文は節に展開することができる．文から節への展開は，文の種別に応じ，次のように再帰的に行う．

（1） 代入文，**goto** 文および標準手続き呼び出し文

これらは文を入れ子に含まないので，文の再帰的な節展開における初期設定に相当する．**図 1**(a)の代入文の場合で示すように，文そのものを節とし，代入節，**goto** 節あるいは標準手続き名を冠して *new* 節などと呼ぶ．ただし，図1では節を2重枠で囲み，未展開の文と区別している．さらに，同図で $adr(v)$ は変数 v に割り当てられた領域の番地計算を，$est(e)$ は式 e の評価を意味するものとする．

（2） 複合文および **with** 文

図1(b)および(c)に示すように，複合文の複合節は入れ子に含む文を代表するだけであり，**with** 文の **with** 節はレコード型となる変数を処理する．

（3） **if** 文および **case** 文

図1(d)に **case** 文の場合を示すが，**if** 文は分岐先が2つある特別な場合に相当する．**if** 節および **case** 節では式の評価を行い，分岐先を決める．

（4） **while** 文，**repeat** 文および **for** 文

図1(e)，(f)および(g)にそれぞれの節展開を示すが，いずれも2つの節を設け，先頭の節は初回の繰

* INSIDERでは，プログラム本体，手続きおよび関数を，必要に応じ区別して扱っているが，実行制御に関しては同一に処理している．そこで，以後特に断らない限り，手続きといえばプログラム本体および関数も対象にしているものとする．

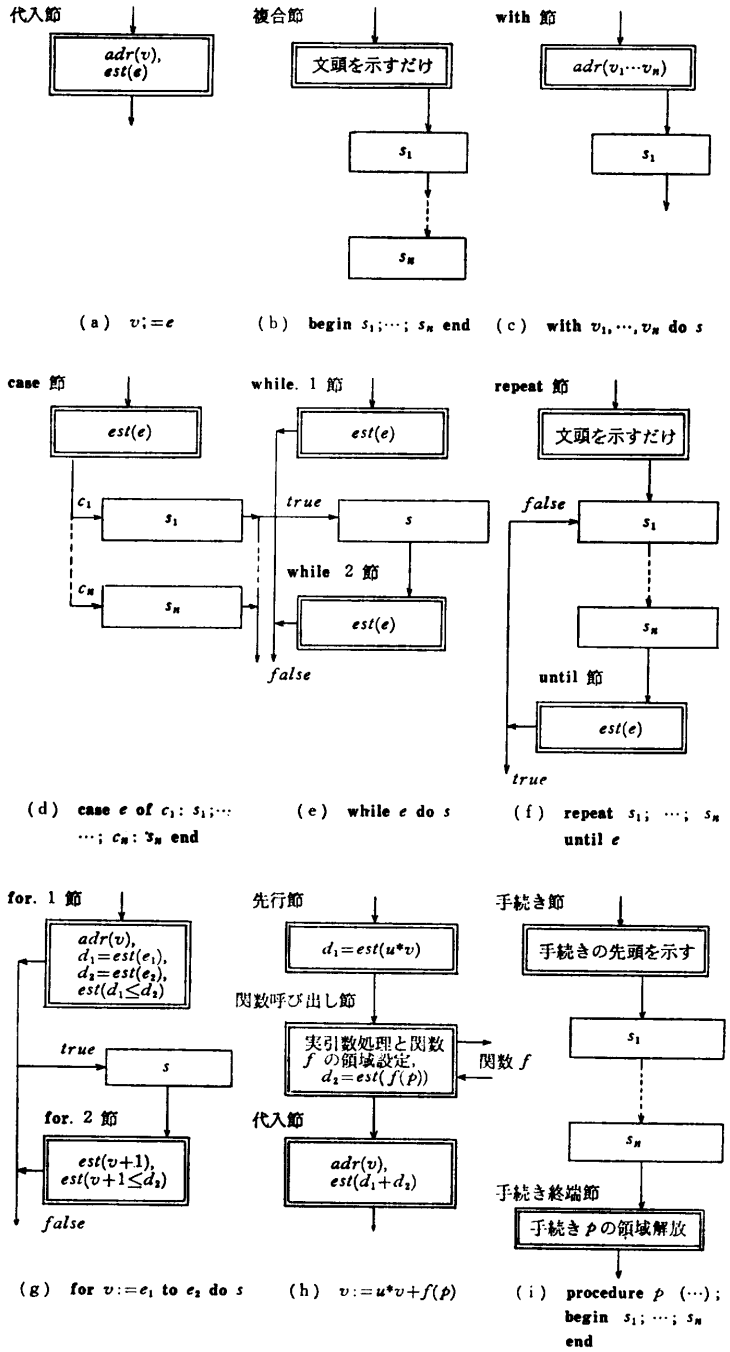

図 1 節による文の枠組の設定

Fig. Formation of framework of statements by nodcs.

り返しの判定を受け持って文頭の節を示し，後尾の節が2回目以降の繰り返しの判定を受け持つ．

（5） 関数呼び出しおよび手続き呼び出し文

今まで述べてきた節で処理する式中に関数呼び出しがあった場合，図1（h）に示すように，関数呼び出し以前の前処理を行う先行節，実引数処理と関数呼び出しのための処理を行う関数呼び出し節，および関数から復帰後の後処理を行う節に3分割し，最後の節にもとの節の名称を継承させる．さらに，実引数に関数呼び出しを含む場合には関数呼び出し節を，また復帰後に新たな関数呼び出しがある場合には後処理を行うべき節を3分割する．これを再帰的に適用して，**図2**の例で示すように，任意の関数呼び出しを含む式を節に展開することができる．手続き呼び出し文は関数呼び出し節に相当する手続き呼び出し節を設ければよい．

（6） ブロックの実行本体

図1（i）に手続きの実行本体の場合を示すが，実行本体の先頭に手続き節を，末尾に手続き終端節を設け，それぞれ手続きの実行のための前処理および後処理を受け持たせる．関数に対しては関数節と関数終端節を設けて手続きと同様な処理を受け持たせるが，プログラム本体に対してはプログラム節およびプログラム終端節を設けて，それぞれプログラム全体の実行のための前処理および後処理を受け持たせる．なお，実行制御の観点からは，ブロックの実行本体を1つの文と考える．

以上の節展開を入れ子に含む文に再帰的に適用し，プログラム全体を節に展開する．なお，ある文の枠組として設定された節は，文の実行制御を行うコマンドにおいて，その文全体を指すためにも用いる．

2.2 INSIDER の支援機能

表1 に INSIDER が備えているコマンドの種別とその機能の概略を，**図3**に各コマンドの構文規則を示す．ただし，同図で[ ]は省略可能を，{ }は0回以上の繰り返しを意味する．〈*node position*〉は，節をソースプログラムと対応づけたときの行番号と，同一行内で節が設定された順番を意味する．〈*qualified identifier*〉は手続き名を指示するためのもので，入れ子構造で宣言された手続きを示すため，上位レベル

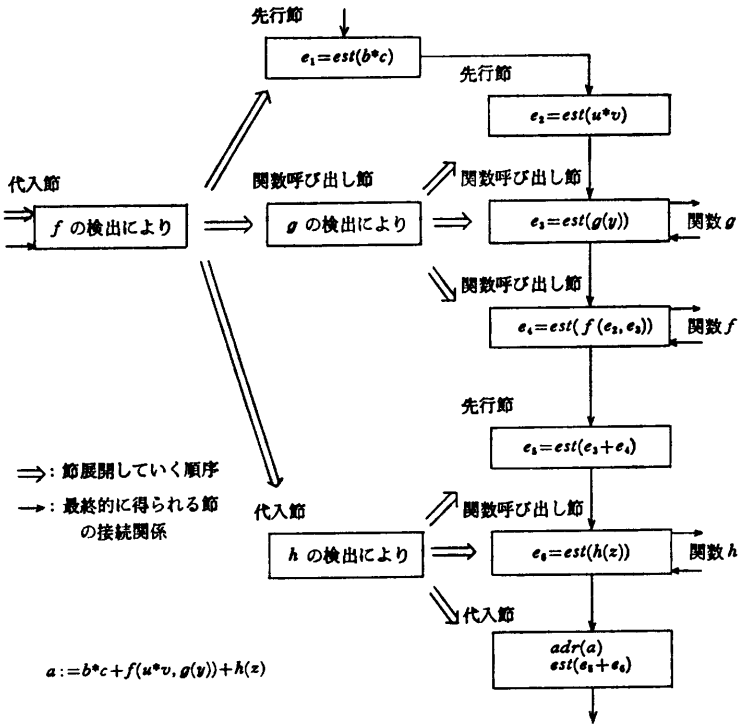

図 2　関数呼び出しを含む代入文の節展開の例

Fig. 2　Example of node expansion of assignment statement containing function calls.

の手続きから点で区切って表わすが，あいまいさが生じないときには上位レベルの手続き名を省くことができる．そのほかの図3で未定義の超変数はすべてPASCAL の構文規則に準拠している．

コマンドを便宜上，指示コマンド，設定コマンド，実行コマンドおよび診断コマンドに大別する．

（1）**指示コマンド**　source コマンドは虫取りの準備を指示するものであり，INSIDER は対象とするプログラムを翻訳して節に展開し，現行節をプログラム全体の先頭であるプログラム節に初期設定する．ただし，現行節とは次に実行すべき節を指す．bye コマンドは虫取りの終了を指示するもので，INSIDER は制御を TSS に戻す．jump コマンドは現行節と同じ手続き内で，かつ現行節を枠組とする文を並置関係または入れ子としてもつ文の先頭の節へ，現行節を変更するものである．

（2）**設定コマンド**　break コマンドは任意の節に実行中断の条件を論理式で

```
⟨node position⟩=⟨unsigned integer⟩ [.⟨digit⟩]
⟨qualified identifier⟩=⟨identifier⟩ {.⟨identifier⟩}
⟨source⟩=SO
⟨bye⟩=BYE
⟨jump⟩=J [⟨node position⟩]
⟨break⟩=B ⟨node position⟩ [.] ⟨Boolean expression⟩;
⟨break cancel⟩=BC ⟨node position⟩
⟨display⟩=D ⟨node position⟩ {[,] ⟨identifier⟩}
⟨display cancel⟩=DC ⟨node position⟩ {[,] ⟨identifier⟩}
⟨recovery node⟩=RVN ⟨qualified identifier⟩ {[,] ⟨qualified
                identifier⟩}
⟨recovery procedure⟩=RVP ⟨qualified identifier⟩ {[,]⟨qualified
                identifier⟩}
⟨recovery cancel⟩=RVC ⟨qualified identifier⟩ {[,] ⟨qualified
                identifier⟩}
⟨run⟩=R [⟨unsigned integer⟩]
⟨continue⟩=C [⟨unsigned integer⟩]
⟨step⟩=S [⟨unsigned integer⟩]
⟨advance⟩=A
⟨execute⟩=E ⟨statement⟩;
⟨recovery⟩=RV
⟨list⟩=L ⟨variable⟩;
⟨frequency⟩=F [⟨qualified identifier⟩]
⟨where⟩=W [⟨node position⟩]
⟨trace nodes⟩=TN [⟨unsigned integer⟩ [⟨unsigned integer⟩]]
⟨trace procedures⟩=TP [⟨unsigned integer⟩ [⟨unsigned inte-
                ger⟩]]
```

図 3　コマンドの構文規則

Fig. 3　Syntactic constructions of commands.

表 1　コマンドの機能の概要

Table 1　Summary functions of commands.

分類	コマンド名	省略名	機能
指示コマンド	source	SO	プログラムの翻訳，節展開等の準備
	bye	BYE	虫取りの終了，制御を TSS に戻す
	jump	J	現行節の変更
設定コマンド	break	B	指定した節に中断条件を設定
	break cancel	BC	指定した節から中断条件を削除
	display	D	変数の動的表示を指示
	display cancel	DC	変数の動的表示の解除
	recovery node	RVN	文の構造に応じて復元する手続きを指定
	recovery procedure	RVP	実行本体の先頭に復元する手続きを指定
	recovery cancel	RVC	復元指示の解除
実行コマンド	run	R	プログラムの先頭から終りまで実行
	continue	C	現行節から終りまで実行
	step	S	1つの文の実行
	advance	A	1つの節の実行
	execute	E	オペランドで指示された文の実行
	recovery	RV	最も近くの復元可能な節まで戻る
診断コマンド	list	L	変数のデータ構造に応じた表示
	frequency	F	節の実行回数の表示
	where	W	節に対応するソーステキストの表示
	trace nodes	TN	実行してきた節の履歴の表示
	trace procedures	TP	呼び出された手続きの履歴の表示

設定する．設定された節では中断条件もあわせて評価し，条件が成立していれば次に実行すべき節を現行節に設定して実行を一旦中止する．この条件設定を解除するのが break cancel コマンドである．display コマンドは指定した代入文が実行されるたびごとに右辺の結果を逐次表示することを指示するもので，代入文を節の位置で指定することもできるが，変数識別子を指定し，その識別子を最左端にもつ変数を左辺とする代入文をすべて指定することもできる（図8の例で，(＊9＊) で示すコマンドとその振舞いを参照）．この表示設定を解除するのが display cancel コマンドである．recovery procedure および recovery node コマンドは，プログラムの状態を以前の状態に復元するため，局所的変数を割り当てるスタック領域と，動的変数を割り当てるヒープ領域の，両データ領域の復元を行う対象とする手続きを指定する．前者は手続きへ入ってきた直後の状態に復元すること（RVP モードと呼ぶ）を指示するのに対し，後者は現行節を枠組とする文と並置された直前の文か，さもなければそれを入れ子にもつ文の先頭に入ってきた直後の状態に復元すること（RVN モードと呼ぶ）を指示する．ただし，端末指定をするように決められているファイル変数 *input* および *output* 変数は復元対象から除く．この復元設定を解除するのが recovery cancel コマンドである．

（3） **実行コマンド** run および continue コマンドはプログラムの終りまでの実行を指示するが，前者が現行節にかかわりなくプログラム節から実行するのに対し，後者は現行節から続行する．step コマンドは現行節を枠組とする文の実行を指示するものであり，呼び出された手続きを含めた動的な入れ子構造を包含した文の並置構造に対する実行を制御する．もし，実行対象となる文の内部に，その文を飛び出す **goto** 文があると，その飛び先を現行節にして実行を中断する．なお，上記3つのコマンドの実行中に，打ち切り時間超過*，実行中断条件成立，あるいは端末からの割り込みがあった場合には，次に実行すべき節を現行節にして実行を中断する．advance コマンドは現行節のみの実行を指示するものであり，文の入れ子構造に対する実行を制御する手段となる．これら4つのコマンドの実行中に異常事態が検出されると，見かけ上検出された節の実行直前の状態で異常の種別を表示して中断するので，**nil** を値にもつポインタ変数で参照したときや，4則演算実行時の異常，添字範囲の逸脱等が発生直前の状態で通知される．

execute コマンドは端末から PASCAL の構文規則に準拠した1つの文を与えて実行させるものであり，現行節の直前に文を一時的に挿入したことに相当する．もちろん，手続き呼び出しを含むような文であってもよい．これにより，実行途中の任意の箇所で変数の診断・変更等ができる．

recovery コマンドは現行節から最も近くにある復元可能な箇所へ復元するものである．まず，現行節を含む手続きに復元指定がなされていると，RVP モードならば手続き節へ，RVN モードならば step コマンドとは対称的に，現行節を枠組とする文と並置された直前の文，さもなければそれを入れ子にもつ文の先頭となる節へ復元する．一方，現行節が復元指定をもたない手続きにいる場合には，手続き呼び出しによる動的な入れ子構造をさかのぼり，指定がなされている手続きを見つけて，復元モードに応じた復元を行う．もちろん，手続き呼び出し文の実行後にその文を復元することは，手続きを呼び出す以前の状態に復元することになる．なお，**goto** 文があるとプログラムの構造が乱されるのであるが，**goto** 文による飛び先を並置構造上と，動的な入れ子構造の上位方向にのみ制限することにより，次のように対処する．すなわち，名札が定義されている文から RVN モードで復元するときには，**goto** 文による分岐が起りうる以前の状態にまで戻るため，並置されている文がそれ以前にあってもそれらを飛び越し，名札定義の文を入れ子にもつ文の先頭の節にまで戻って復元する．

（4） **診断コマンド** list コマンドは指定された変数の静的およびポインタ変数による動的なデータ構造を，各データ型の様式に従って表示する．特にポインタ型に対しては，動的変数に一連番号を付し，それらの参照関係を含めて動的変数の内容を表示する（図8の(＊15＊)および(＊31＊)のコマンドによる表示例を参照）．この参照関係はポインタの値が **nil** になるまで続ける．frequency コマンドは指定された手続き内の各節の実行回数を，where コマンドは指定された節が見かけ上処理対象とするソーステキストを表示する．また，trace procedures コマンドは過去に呼び出された手続きの，trace nodes コマンドは過去に実行した節の履歴を，いずれも現在から逆にたどって表示する．

* これらのコマンドのオペランドで，CPU 時間による打ち切り時間を指示することができる．実行時間の見積りに関しては 4.2 節参照．

3. 内部処理方式

3.1 システム構成と実行制御

INSIDERは，OSと情報の授受をするためにアセンブラ言語で書かれた外部手続きを除き，PASCALで記述されており，そのシステム構成を**図4**に示す．INSIDERはPASCALコンパイラを1つの手続きとして内蔵しており，コンパイラは虫取りの対象となるプログラム，およびPASCALの構文規則による文，式あるいは変数を書くことが許されているコマンドのオペランドを，節単位で直接実行可能な機械語モジュールに翻訳する．その際，各モジュールの独立性を保つため，レジスタ類は1つのモジュール内でのみ閉じた使い方をしている．コンパイラは翻訳と同時にプログラムを並置構造と入れ子構造に基づいて節に展開し，ネットワーク構造からなる節管理表を作成する．そこでは節ごとに，構造を指示する結合情報，機械語モジュール，節に関連したソーステキスト，動的表示のための機械語モジュール，設定された中断条件とその機械語モジュール，その他診断情報等を集約している．これらはすべてINSIDER側のデータ領域の，主としてヒープ領域に動的に作成する．一方，虫取りの対象となるプログラムのためのデータ領域は，INSIDER側とは独立に設定するので，お互いに相手の領域を意識する必要がない．

各節が受け持つ機械語モジュールは，スタック領域の処理を行う手続き呼び出し節，手続き節，手続き終端節等を除き，変数に割り当てられた領域の番地計算と式の評価のみを任務としており，したがってデータ領域を参照するだけであって，変数の値の更新および計算の流れの制御は節実行制御部が行う．代入節，*new*節等の，変数の値を更新するような節では，節実行制御部が機械語モジュールから番地と更新すべき値を受け取り，復元の指定があるときには，次節で述べる待避処理を施してから更新処理を行う．このほか，節実行制御部は指示に応じて，動的表示あるいは中断条件のための機械語モジュールを駆動し，さらに割り込み，打ち切り時間超過および異常事態発生でOSからの割り込み等で設定されるフラグを点検し，中断の有無を決定する．

3.2 復元処理

プログラムの実行状態を**2**章で述べたように，プログラムの構造に従って復元するためには，文頭の節を実行するとき，あらかじめすべての変数の値を待避しておけばよいのであるが，これでは待避のために莫大な記憶容量を必要とし，実際的ではない．そこで，ある変数の値の更新があったときには，設定された復元モードに応じて必要な箇所に待避データを埋め込んでいく方式を考える．**図5**はデータの待避箇所と復元経路の例を模式的に示したものである．ただし，同図では字下りで入れ子構造を表わし，各手続きが注釈に示すような復元モードに設定されており，現行節は手続きの多重呼び出しにより手続きrにあって，動的な入れ子構造を構成しているものとする．いま，手続きrの

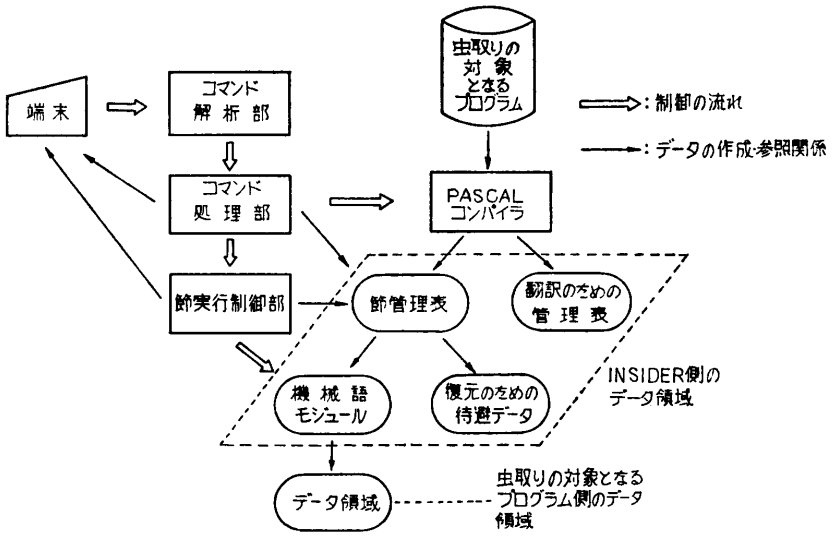

図4 INSIDERのシステム構成
Fig. 4 System configuration diagram of INSIDER.

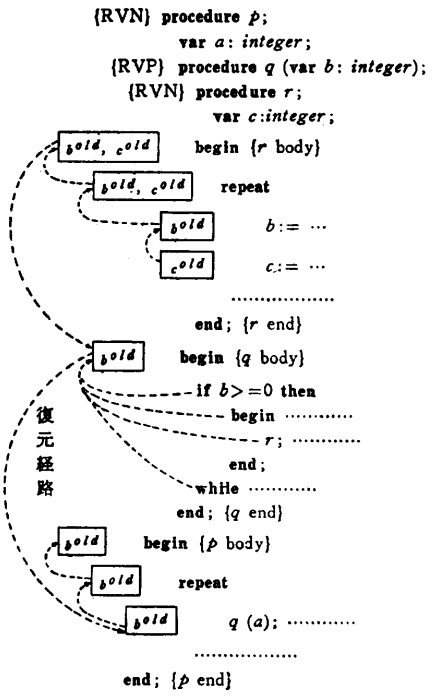

図 5 待避箇所と復元経路の例
Fig. 5 Example of saved points and recovery paths.

局所的変数 c への代入文を考えると，r がRVNモードに設定されているので，これ以降の r 内の任意の箇所からのプログラム構造に基づく復元要求に応じるため，図5の c^{old}→で示すように，自身の節を含め，その代入文を入れ子に含む r 内のすべての文の先頭の節管理表に更新前の値を待避する．一方，手続き r では大域的変数である変数 b への代入文を考えると，b は引数を通して手続き p の局所的変数 a でもあるので，手続きの呼び出しを通じてその代入文を動的に包含している文をたどり，図5の b^{old}→で示すように，各手続きの復元モードに応じて b の更新前の値を待避する．ただし，待避中に同一変数に対する待避データが見つかると，それ以上の入れ子のレベルにはすでにその変数に関する待避データが格納ずみであるので，そこでデータの待避を打ち切る．図5には任意の箇所からの復元経路もあわせて示してある．図6には節管理表に格納する待避データのためのデータ構造を示すが，復元先の先頭番地，データ長およびデータ本体を1組とし，出現順にスタックしていく．さらに手続きの再帰的呼び出しに対処するため，このスタック構造をスタックする2重構造となっている．待避データはスタック領域あるいはヒープ領域の区別なく，一律に作成する．なお，復元モードに対応して復元箇所となる節では，入れ子に含む内部の文からの待避データの格納要求に備え，最上部のスタックを初期化する．

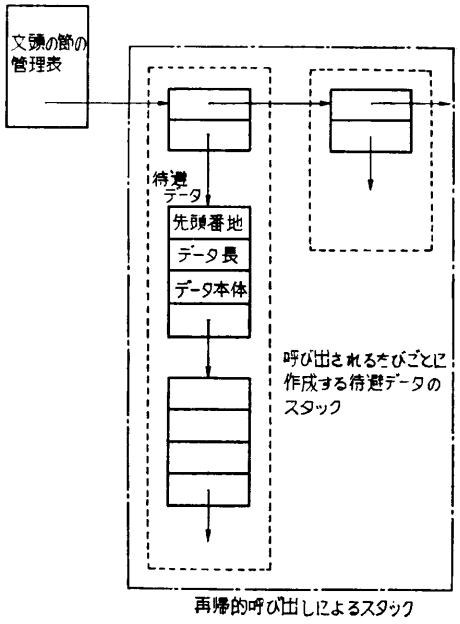

図 6 2重スタックからなる待避データ
Fig. 6 Saved data formed by double stack.

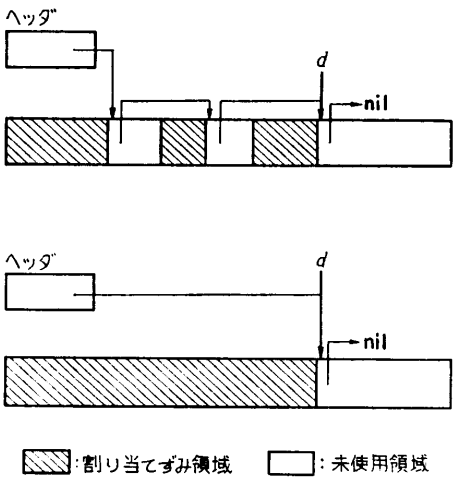

図 7 ヒープ領域の復元
Fig. 7 Recovery of heap area.

ファイルの入出力に対しても代入節と基本的には同じ機構で対処できるが，復元すべきデータそのものはファイルに保存されているので，復元に必要なファイル情報を待避する．ただし，ファイルは入力または出力のいずれかのモードでのみ使われるものとする．

スタック領域の復元は手続きの呼び出しによる結合をたどり，各手続きのための領域を解放していけばよ

```
ALLOC DD(INPUT) DS(*)
READY
ALLOC DD(OUTPUT) DS(*)
READY
ALLOC DD(PASIN) DS(EIGHT.DATA)
READY
CALL DEBUGGER(INSIDER)
  INSIDER IS STARTED
?(*1*) SO
  PASCAL PROGRAM SOURCE LIST
     1 PROGRAM EIGHTQUEEN(OUTPUT);
     2  TYPE LISTP=@LIST;
     3       LIST=RECORD KEY:INTEGER;
     4                   NEXT:LISTP
     5            END;
     6  VAR I:INTEGER; Q:BOOLEAN;
     7      A:ARRAY[ 1.. 8] OF BOOLEAN;
     8      B:ARRAY[ 2..16] OF BOOLEAN;
     9      C:ARRAY[-7.. 7] OF BOOLEAN;
    10      X:LISTP;
    11
    12  PROCEDURE TRY(I:INTEGER;
    13    VAR Q:BOOLEAN; VAR P:LISTP);
    14   VAR J:INTEGER;
    15  BEGIN J:=0; NEW(P); P@.NEXT:=NIL;
    16   REPEAT J:=J+1; Q:=FALSE;
    17    IF A[J] AND B[I+J] AND C[I-J]
    18     THEN
    19      BEGIN P@.KEY:=J;
    20       A[J]:=FALSE; B[I+J]:=FALSE;
    21       C[I-J]:=FALSE;
    22       IF I<8 THEN
    23        BEGIN TRY(I+1,Q,P@.NEXT);
    24         IF NOT Q THEN
    25          BEGIN A[J]:=TRUE;
    26           B[I+J]:=TRUE;
    27           C[I-J]:=TRUE
    28          END
    29        END
    30       ELSE Q:=TRUE
    31      END
    32    UNTIL Q OR (J=8)
    33  END;  (*TRY*)
    34
    35 BEGIN
    36  FOR I:= 1 TO  8 DO A[I]:=TRUE;
    37  FOR I:= 2 TO 16 DO B[I]:=TRUE;
    38  FOR I:=-7 TO  7 DO C[I]:=TRUE;
    39  TRY(1,Q,X);
    40  IF Q THEN
    41   WHILE X<>NIL DO
    42    BEGIN WRITE(X@.KEY:4); X:=X@.NEXT END;
    43  WRITELN
    44 END.
  NO ERRORS IN THIS PASCAL PROGRAM
    33.1   PROGRAM-NODE
?(*2*) R
   1   5   8   6   3   7   2   4
    33.1   PROGRAM-NODE
?(*3*) RVP TRY
?(*4*) A
    36.0   FOR.1-NODE
?(*5*) S
    37.0   FOR.1-NODE
?(*6*) S
    38.0   FOR.1-NODE
?(*7*) S
    39.0   PCALL-NODE
?(*8*) A
    15.0   PROCEDUR-NODE
?(*9*) D # A,B,C
?(*10*) B 22, I=3;
?(*11*) C
     20.0:  A[  1] <--     FALSE
     20.1:  B[  2] <--     FALSE
     21.0:  C[  0] <--     FALSE
     20.0:  A[  3] <--     FALSE
     20.1:  B[  5] <--     FALSE
     21.0:  C[ -1] <--     FALSE
     20.0:  A[  5] <--     FALSE
     20.1:  B[  8] <--     FALSE
     21.0:  C[ -2] <--     FALSE
  BREAK BY CONDITION:  I=3;
    23.0   BEGIN-NODE
?(*12*) BC 22
?(*13*) DC # A,B,C
?(*14*) E WRITELN(I,J,A[I]);
        3          5     FALSE
?(*15*) L X;
 #0 --> #1
 #1.KEY:          1
 #1.NEXT --> #2
 #2.KEY:          3
 #2.NEXT --> #3
 #3.KEY:          5
 #3.NEXT --> NIL
?(*16*) RV
    15.0   PROCEDUR-NODE
?(*17*) L I;
        3
?(*18*) RV
    15.0   PROCEDUR-NODE
?(*19*) L I;
        2
?(*20*) C
   1   5   8   6   3   7   2   4
    33.1   PROGRAM-NODE
?(*21*) TR 3
    15.0   PROCEDUR-NODE
    15.0   PROCEDUR-NODE
    15.0   PROCEDUR-NODE
?(*22*) TN 5
    44.0   PROGEND-NODE
    43.0   WRITE-NODE
    41.1   WHILE.2-NODE
    42.2   ASSIGN-NODE
    42.1   WRITE-NODE
?(*23*) F TRY
   NODE-ID    NODE-KIND     FREQUENCY
    15.0   PROCEDUR-NODE     115
    15.1   ASSIGN-NODE       115
    15.2   NEW-NODE          115
    15.3   ASSIGN-NODE       115
    16.0   REPEAT-NODE       115
    16.1   ASSIGN-NODE       884
    16.2   ASSIGN-NODE       884

?(*24*) RVN TRY
?(*25*) B 21, (I=1) AND (J=1);
?(*26*) R
  BREAK BY CONDITION:  (I=1) AND (J=1);
    22.0   IF-NODE
?(*27*) W
    22.0   IF-NODE
  IF I<8 THEN
?(*28*) RV
    21.0   ASSIGN-NODE
?(*29*) RV
    20.1   ASSIGN-NODE
?(*30*) RV
    20.0   ASSIGN-NODE
?(*31*) L A;
 [1]:       TRUE
 [2]:       TRUE
 [3]:       TRUE
 [4]:       TRUE
 [5]:       TRUE
 [6]:       TRUE
 [7]:       TRUE
 [8]:       TRUE
?(*32*) J 16
    16.0   REPEAT-NODE
?(*33*) RVC TRY
?(*34*) C
   2   4   6   8   3   1   7   5
    33.1   PROGRAM-NODE
?(*35*) BYE
  INSIDER IS TERMINATED
READY
```

図 8 「8人の女王」のプログラムの実行制御例

Fig. 8 Example of execution control to the eight queens program.

い．一方，ヒープ領域は *dispose* による領域解放による空き領域を 図 7(a) に示すように，リスト構造で管理しているが，復元箇所の節の実行に先立って，その時点で未使用のヒープ領域の先端の番地 d を待避しておき，同図(b)に示すような割当て状態に復元する．

3.2 コマンドの処理

ここでは最も錯綜した処理を要するいくつかのコマンドについて概説する．execute コマンドに対しては，オペランドで指示された文を翻訳して節展開した上で，ほかの実行コマンドと同様に節実行制御部の制御のもとで実行させる．list コマンドに対しては，まずオペランドで指示された変数を翻訳して，その変数に割り当てられた領域の先頭番地を計算する機械語モジュールを作成し，同時に変数のデータ型を求める．次にそのモジュールを実行させて変数の先頭番地を求め，その番地以降に格納されているデータを，先に得られているデータ型を手本として，データ型に応じた様式で出力する．recovery コマンドに対しては，現行節から最も近くにある復元可能な節を求め，その節管理表が所持している最上位のスタックの待避データをもとにして，データ領域およびファイルの復元を行う．待避データをスタック時とは逆方向に取り出して復元していくので，レコードの可変フィールドや *dispose* による領域の多重使用に対しても書替えを重ねることにより，結局復元箇所で有していた状態に復元される．

なお，INSIDER の処理系の中で，番地指定に基づくデータの参照および格納，割り込み処理，打ち切り時間設定，データ領域の設定および解放，機械語モジュールの駆動等はアセンブラ言語による外部手続きで実現している．

4. INSIDER の使用例と性能評価

4.1 「8人の女王」のプログラムによる使用例

図 8 は「8人の女王」のプログラム[12] を例に取り上げ，故意にポインタ変数を使うように改修した上で INSIDER の機能を紹介したものである．同図の冒頭はデータセットの割当てと，INSIDER の起動のための TSS コマンドであり，"?" で始まる行が INSIDER へのコマンドで，注釈は説明上挿入したものである*．

* コマンドの字句解析はコンパイラに委ねているので，コマンドの中に注釈を挿入することができ，注釈を介してコンパイラ・オプションの指定もできる．

表 2 「8人の女王」のプログラムによる実行時間の比較
Table 2 Comparison of running time by the eight queens program.

場合分け				実行時間 (ms)	節当りの平均実行時間 (μs)	倍率
通常のコンパイラの翻訳による実行				5.6	0.98	1
INSIDER の制御下で実行	機械語モジュール側で更新処理も行わせた場合			170	30	30
	復元指定なしで実行			220	39	39
	復元指定をして実行	プログラム本体	手続き TRY			
		—	RVP	410	72	73
		RVP	RVP	420	74	73
		RVN	RVP	430	75	77
		—	RVN	920	161	164
		RVP	RVN	920	161	164
		RVN	RVN	930	163	166
		RVP	—	440	77	79
		RVN	—	700	123	125

主な点のみを述べると，(＊3＊) で手続き TRY を RVP モードに設定し，(＊16＊) の recovery コマンドで手続きの先頭に，(＊18＊) の再度の recovery コマンドで再帰的に1つ前に呼び出されていた手続きの先頭に復元している．(＊23＊) のコマンドに対しては割込みで表示を打ち切り，(＊24＊) からの一連のコマンドでは RVN モードのもとで復元を行わせ，jump コマンドで計算の流れをかえて別解を求めている．

4.2 使用例に基づく性能評価

INSIDER は現在 HITAC-M 200 H を CPU とする VOS 3 システムのもとで稼動している．表 2 は前節のプログラムを用い，復元モードの設定をかえて実行時間の比較を行ったものである．この例では手続きの再帰的呼び出しを行っており，復元機構から見ると大域的変数および引数を介しての変数の待避のため，厳しい条件となっている．INSIDER のオーバヘッドは，復元指定のないときには一定であるが，復元指定があるとプログラムの規模，特にデータ領域の規模が大きくなるにつれ増大する．INSIDER は復元指定のないときで節当り約 150 バイトの領域を必要とするので，これから制御できるプログラムの大きさが決まる．ただし，復元指定があれば，プログラムにより大幅に異なるが，一般に大量の記憶容量を必要とするので，重点とする手続きを指定するなどの配慮が必要である．

5. あとがき

プログラムに誤りがあることを知るのは実行結果を得てからであり，その結果から誤りの原因を推定しなければならない．INSIDERのもとでは，利用者は自身の制御のもとでプログラムを「行きつ戻りつ」させながらその挙動を観察することができ，原因の究明を効率的に行うことができるばかりでなく，プログラムを理解・診断する有益なツールにもなりうる．

現在稼動中のINSIDERは，誤りの修正をTSS下のエディタに委ねているが，INSIDER自身の中で構造的な編集機能[13]を行使できるようにすべく拡張中である．

謝辞　北海道大学竹村伸一教授および田川遼三郎教授には本研究を進めるに当り貴重なご教示をいただき，また桃内佳雄助手には本論文をまとめるに当り懇切なご意見をいただいた．ここに記して深謝申し上げます．

参考文献

1) 鳥居宏次，二木厚吉，真野芳久：プログラミング方法論の展望，情報処理，Vol. 20, No. 1, pp. 22-43 (1979).
2) Sandewall, E.: Programming in an Interactive Environment : The Lisp Experience, ACM Computing Surveys, Vol. 10, No. 1, pp. 35-71 (1978).
3) 和田英一：ソフトウェア製造ツールの最近の傾向，情報処理，Vol. 20, No. 8, pp. 681-68 (1979).
4) 黒川利明：Lispシステムにおけるデバッギング・ツール，情報処理，Vol. 20, No. 9, pp. 809-819 (1979).
5) Conway, R. W. and Wilcox, T. R.: Design and Implementation of a Diagnostic Compiler for PL/I, Commun. ACM, Vol. 16, No. 3, pp. 169-179 (1973).
6) 原田賢一，Zelkowitz, M. V.：インタラクティブPL/Iシステムの設計と作成，情報処理，Vol. 16, No. 2, pp. 85-92 (1975).
7) 藤村直美，牛島和夫：FORTRANプログラムの動的解析システムとプリコンパイラ，情報処理，Vol. 17, No. 6, pp. 547-550 (1976).
8) 春原　猛，大井房武，関本彰次，中村敏行：高位言語デバッギングシステムSOLDA，情報処理，Vol. 20, No. 5, pp. 405-411 (1979).
9) Hart, J. J.: The Advanced Interactive Debugging System (AIDS), SIGPLAN Notices, Vol. 14, No. 12, pp. 110-121 (1979).
10) Cichelli, R. J. : Pascal-I—Interactive, Conversational Pascal-S, SIGPLAN Notices, Vol. 15, No. 1, pp. 34-44 (1980).
11) Chu, Y. ed.: High-Level Language Computer Architecture, p. 273, ACADEMIC PRESS (1975).
12) Wirth, N. : Algorithms+Data Structures=Programs, p. 366, PRENTICE-HALL (1976).
13) 宮本衛市，浅見可津志：プログラム構造に基づいた編集機能をもつテキスト・エディタ，情報処理学会論文誌，Vol. 20, No. 6, pp. 474-480 (1979).

(昭和55年4月17日受付)
(昭和56年3月19日採録)