記号処理 18－2
（1982. 3. 25）

PETL： 知識を集積するための プログラミングシステム

塚本享治
（電子技術総合研究所）

熊谷毅
（宇都宮大学）

1. まえがき

知識工学のテーマは，「知識」の表現方法，獲得方法，蓄積方法と利用方法である。ここで，「知識」を具体的な「プログラム」で置きかえると，本稿に述べるプログラミングシステム Petl のテーマとなる。

Petl システムは，Lisp に様々な機能を盛り込んだ会話型のプログラミングシステムである。本稿では多くの機能のうちでも特に特徴的な，知識としてのプログラムを集積するためのデータベース機能と，それを確実に利用できるようにするための支援機構について述べる。

2. データベース

Petl システムと外部を結ぶ入出力ストリームの1つとしてキーインデクスファイルが使用できる。キー（見出し）によってレコードの位置が指定でき，データベースともいう。検索の効率や管理のやり易さの点で，プログラムやデータの蓄積に適している。

2.1 データベースの構造

Petl システムのデータベースは4つの領域から成る。

(1) 属性領域　システムがデータベースの管理のために使用し，データベース作成者名，データベースのタイプ等を格納する。プログラムからはアクセスできない。

(2) ケイパビリティ（CAP）領域　データベース作成者がデータベースの管理のために使用し，作成者以外のアクセスは許されない。ただし，システムの用意した一部の組込関数には読むことを許す。特に，キー W と R にはそれぞれデータベースに対するリードおよびライトを許す者の名前を登録する。

(3) データ辞書（DD）領域　読み書きの許された利用者が，自由に利用できる領域。データ領域の管理に用いる。

(4) データ（DB）領域　読み書きの許された利用者が自由に利用できる領域。素データを格納する。

CAP 領域と DD 領域は S 式を格納するのに対し，DB 領域はデータベースのタイプで定まる型のレコードを格納する。3つの領域はそれぞれ独立であり，各領域に同名のキーを使用することが可能である。

2.2 キー

すべてのアトムをキーとすることができる。キーにはデータ型が付与され，各型のアトムはそれぞれ独立なキーとなる。また，組込関数 HASH を使用することにより，キーを一般の S 式に拡張することが可能である。長大な S 式が少量のバッファで入出力できるようにするために，キーには特殊なコーディングが施され，1つの論理的なキーが最大 256 個の物理的なキーに対応する。

2.3 データベースのタイプ

データベースのタイプは，DB 領域に格納するレコードの形式を定める。これは，データベース作成時に，関数 OPEN で指定し，属性領域に登録される。データベースのタイプには次の4種類がある。

論理的キーに対応するレコードが1つの S 式から成る EXPR 型，同じく任意の文字列から成る TEXT 型，レコー

ドが2進データであるBIN型と，2進データのうち特に線画図形であるところのFIG型である。

2.4 データベースを扱う関数

データベースを扱う関数は6種類に分けられる。すなわち，追加・更新関数，参照関数，削除関数，キー検索関数，キー逐次読取関数，そして位置決め関数である。関数名はそれぞれ，PUT, GET, REM, KEY, MAP, POSで始まり，対象となる領域を示す名前CAP, DD, DBが続く。以下にDB領域を扱う関数について示す。他の領域に関してもほぼ同様である。

(1) データベース追加更新関数
(PUTDB db sexp key)

データベースdbのDB領域にキーkeyで値sexpを書き込む。keyが存在する場合は更新，存在しない場合は追加される。

(2) データベース参照関数
(GETDB db key [array-name])

データベースdbのDB領域からkeyで指定したキーの値を読む。タイプがBINまたはFIG型の場合，配列array-nameに読んだ内容を書き込む。

(3) データベース削除関数
(REMDB db key)

データベースdbのDB領域からキーkeyとその内容を削除する。

(4) データベースキー検索関数
(KEYDB db key)

データベースdbのDB領域にキーkeyが存在するかどうか検査する述語関数である。

(5) データベースキー逐次読取関数
(MAPDB function db key)

データベースdbのDB領域のキーkeyから順にキーを読み取り，それを1変数関数functionに適用する。先頭のキーから処理を行う場合にはkeyに'BOFを指定する。キーは16進整数キー，10進整数キー，実数キー，ストリングキー，リテラルアトムキーの順に並んでおり，各タイプにおいては値順または辞書順となっている。

(6) データベースキー位置決め関数
(POSDB db key)

逐次読取関数の評価中に，次に読取るキーの位置をkeyに変更する。特に先頭を指定する場合は'BOFを，最後を指定し処理を打切る場合には'EOFを指定する。

3. プログラムデータベース

プログラムを蓄積し，相互利用するために，前章のデータベースを使って実現されたプログラムデータベースについて述べる。エディタ，コンパイラ等のユーティリティはいずれもこのプログラムデータベースを対象とする。

3.1 プログラムデータベースの構造

1組のプログラムデータベースは，ソースプログラム，オブジェクトプログラム，実行可能コードを専用に格納する3個のデータベースから成る。それぞれ，ソースデータベース，オブジェクトデータベース，ライブラリデータベースと呼び，タイプは各々EXPR, EXPR, BIN型である。ファイル名にはエクステンションとしてそれぞれ(SRC), (OBJ), (LIB)が付く。

プログラムは次に述べるモジュールを単位として作成し，ソースデータベースに格納する。各データベースへのアクセスとデータベース間の変換は，モジュールを単位として Fig 1 の各関数

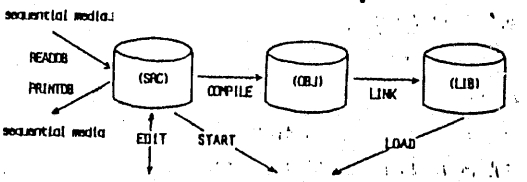

Fig 1. プログラムデータベースと関数の関係

によって行われる。

3.2 モジュール

Lispには，他のプログラム言語にあるような，情報隠蔽の意味でのモジュールやデータタイプそして仕様記述などの概念が欠けている。このことが会話型システムとして極めて有効である反面，大規模なプログラムの開発や，第3者の開発したプログラムの利用にあたっては大きな障害となる。

Pathシステムでは，言語の構造としてでなく，データベースの構造として，次のようなモジュールを実現することによって，これらの欠点を克服している。

モジュールを，プログラムデータベースのDD領域にモジュール名をキーとして格納された仕様記述と定義する。

3種のデータベースに対応し，モジュールの形式も3通りである。それぞれ，ソースモジュール，オブジェクトモジュール，ライブラリモジュールと呼ぶ。ソースモジュール以外のモジュールはソースモジュールをコンパイルしリンクすることによって自動的に作成される。

Fig 2はソースモジュールの形式である。ここで，[,] は繰り返しを意味し，**PROG** の前後のリストはそれぞれの内部において自由な交換が許される。随所に現われるキーは特別な場合を除き，10進整数値であり，DB領域へのポインタとなっている。

```
(MODULE  module-name  date
  [(GLOBAL  [(atom type-def-key comment-key)])]
  [(REFER  [(function-name function-type
            module-name file-name)])]
  [(MACRO  [(macro-name arg-list macro-body)])]
  (COMMENT  atom)
**PROG**
  [(function-name function-type args-list
                vars-list [function-def-key])]
  [(decimal-integer  EXEC  [form-def-key])] )
```

Fig 2. ソースモジュールの形式

各項目を句と呼び，それぞれ次のことを意味する。

(1) GLOBAL句

モジュール内に定義する関数，配列，あるいは大域変数を他のモジュールに提供するために，型定義と関連するコメントを宣言する。関数の場合の型定義は，例えばSETQの場合Fig 3のようになる。この句に記載されない関数，配列や変数はコンパイルすると隠される。

```
(SETQ ["VAR1":atom VAL1:sexp]):FSUBR
  = VALn:sexp
```

Fig 3. 型定義の例

(2) REFER句

外部のモジュールで定義された（GLOBALで宣言された）関数の参照を宣言する。コンパイル時に使用される。

(3) MACRO句

PROG 以降に表われるプログラムの評価と（オープン）コンパイルに必要なマクロを定義する。

(4) COMMENT句

PROG 以降に表われるプログラムにおけるコメントの指示子を定義する。

(5) 関数句

関数を定義する。function-typeとargs-listはその型と引数のリスト，vars-listはゼネレータ型の関数引数の初期値のリストである。function-def-keyは，関数本体の定義を格納するDB領域のキーである。これが複数あるときは，前にあるほど版が新しい。例えば

(DE FACT (N) (COND ---))

は次のような関数句となる。

(FACT EXPR (N) NIL m_1 m_2 ---)

(6) 雑数句

関数定義以外の一般の形式を定義する。10進雑数は識別番号である。

ソースモジュールをコンパイルしさらにリンクすると，再配置可能な実行コードを定義するFig 4の形式をしたライブラリモジュールがライブラリデータベースに登録される。

```
(MODULE  module-name (version date) date
  (GLOBAL  [(latom type-def-key comment-key)])
  (LOCAL   [latom])
  (REFER   [(function-name function-type ...)])
  (COMMENT latom)
 **PROG**
  [(function-name function-type nr-args vals-list
                code-length code-key consts-key)]
  [(decimal-integer code-length code-key consts-key)] )
```

Fig 4　ライブラリモジュールの形式

ここで，GLOBAL句，REFER句，COMMENT句は，キーの値を除いて対応するソースモジュールのものと同じである。

(7)　LOCAL句

ソースモジュールをコンパイルした結果，他のモジュールから隠す必要のあることが判明したリテラルアトムをローダに対し宣言する。

ソースモジュールに記述されていた仕様がこの段階まで保存されている。

3.3　編集，実行，管理

以上に述べたモジュールは，**PROG** 以降の句の順序を除けば，アルゴリズムとは直接には関係のない管理のためのものである。コンパイラで使われるだけでなく，編集や実行の管理，システムの管理，利用者間でのプログラムの共同利用などに非常に重要な役割を果たしている。

エディタでは，ソースモジュールの仕様記述そのものと，仕様記述から指し示された関数や型などの定義本体が編集の対象となる。定義本体の一部を変更したときには新たなキーでDB領域に書き込み，ソースモジュールの対応する場所にそのキーを登録すれば良い。古い版に戻したり，古い版の消去などもリスト操作によって極めて簡単に行える。

ソースモジュール名に対応するDB領域のキー以外のキーは，特殊な例外を除き通し番号の整数である。編集などの結果不要なキーができても，データベースからは削除しないで残しておき，暇をみつけて時折データベース中の必要なキーをマークし残りのキーを削除する。このがーベッジコレクションを行わない限り，誤ってモジュールからキーを削除しても容易に回復することができる。

関数 START によるソースモジュールの実行や，LOAD によるライブラリモジュールの実行も，モジュールの記述に沿って行う。まず，LOCAL 句に従ってリテラルアトムの局所化のための前処理を行い，コメント指示子の変更と MACRO 句で指示された マクロの定義を行う。続いて，**PROG** 以降の関数句と整数句を記述されている順に最新の版を読み込んで定義，ロードまたは評価する。最後に，リテラルアトム局所化のための後処理を行う。

関数 MAPF によって，全ての利用者の全てのファイル名を取り出すことができ，これを用いてアクセスの許可されたデータベースから，目的の仕様に合うモジュールを探し出すこともできる。

4.　支援機構

Lisp処理系はアルゴリズムの開発に用いられることが多く，その場合にはプログラムには常にバグが存在する。また，会話型システムにつきものの操作ミスや端末からのブレイクもいつ発生するかわからない。このような過酷な環境においても，データベースのコンシステンシを保つためには，キーインデクスファイルを導入しただけでは不十分である。　　また，過酷な環境において他人に対しプログラムの動作を保証することも必要である。そのために Pal システムでは以下に示す機構を提供している。

4.1　トラップ処理

プログラムの実行中に発生したエラーに対し，予め定義しておいた関数を暗黙のうちに実行中のプログラムに挿入して評価することができる。このための関数をトラップ処理関数という。有効スコープを定義し，その内側で発生したエラーに対し回復操作を行うために，次のような関数が使用できる。

(TRAP HANDLER:function [FORM$_1$:sexp])

FORMを順に評価しているときにエラーが発生すると，HANDLERに制御が渡り，HANDLERでエラーが修復されると，FORMの評価を再開する。関数TRAPは入れ子状に使用でき，修復が不可能なエラーに対しては，関数RESIGNALによって上位のレベルに定義された処理関数に制御を移すことが可能である。入れ子状態の最上位に位置するのが，システムベクトルに定義されたトラップ処理関数である。

システムは，トラップ処理関数に引数としてエラーの状況を示すコード，エラー発生時のスタックポインタの値，および修復処理に必要なパラメータのリストを渡す。

Fig 5は，未定義の関数を評価しようとしたときと，既にオープンされている入出力ストリームを再度オープンしようとしたときに発生するエラーに対して修復を行い，その他のエラーの場合には上位のトラップ処理関数に修復を任せるトラップ処理関数である。

```
(DE HANDLER (CODE SP TLIST)
  (CASE CODE
        ((111) (; undefined function)
         (COND ((MEMQ '/MYFILE (OPEN)))
               ((OPEN '/MYFILE 'EXPR)))
         (COND ((KEYDB '/MYFILE (CAR TLIST))
                (LIST (EVAL (GETDB '/MYFILE (CAR TLIST)))))
               ((PRIN1 "undefined function: ")
                (PRINT (CAR TLIST))
                (LIST (EVAL (READ))))))
        ((208) (; stream already opened)
         (XV SP 1 NIL))
        (T (RESIGNAL))) )
```

Fig 5　トラップ処理関数の例

4.2 保証処理

プログラムの実行中に発生するエラーやブレイクの処理，および大域制御関数では，実行中の途中の処理を跳ばして上位の処理に戻ることが多い。跳ばされる側のプログラムにしてみれば，そのような過酷な扱いに対しては，その後の処理を保証するのは通常困難である。

Pa1dシステムでは，有効スコープを定義し，その内側で発生した大域制御に対しても処理を確実に保証するために，次のような関数が使用できる。

(ENSURE [FORM$_1$:sexp])

FORMを順に評価しているときに，関数ENSUREを跳びこすような大域制御が発生すると，システムは評価中のFORM$_i$の評価を放棄し，FORM$_{i+1}$以降を評価したのち大域制御を実現する。この保証処理中に大域制御が発生すると，より上位の跳び越し先が新たな跳び越し先となる。関数ENSUREは入れ子状に使用することができる。

保証処理の特別なものとして，入出力ストリームをクローズするときに評価される入出力終了処理関数と，システムの処理を終えるときに評価されるシステム終了処理関数をシステムベクトルに定義することができる。

4.3 リテラルアトムの隠蔽

複数のモジュールを同時に使用する際の関数名，配列名や変数名の競合を避けるために，モジュール内のリテラルアトムを隠蔽することができる。3.2に述べたとおりである。

それを支援するために，オブジェクトリスト上にリテラルアトムを複製する関数と，複製したリテラルアトムを

・オブジェクトリストから外し，見えない位置に移す関数が用意されている。

データベース名を隠すことにより，通常の関数によるアクセスを排除することが可能である。また，既存の関数を隠し，新たに同名の関数を定義することも可能である。

本章に述べた機構をさらに確実なものにするために，システムでは割込の禁止と解除をきめ細く行っている。プログラムからも関数を通して割込の制御を行うことができる。

5. 実用例

Petl システムには，以上に述べた機能のほかにも数多くの機能が盛り込まれている。筆者の周辺における最近の使用例をあげその説明にかえる。

(1) 機械装置の故障診断[1)]

ミンスキのフレーム論を用いた機械装置類の故障診断。典型的な Lisp 的使用例である。

(2) 日本語構文解析

機械翻訳を目指した語の結合関係を利用した日本語構文解析[2)]。プログラム，辞書，ユーティリティがそれぞれ，1，23，2個のデータベースから成るシステムである。本システムの特徴を生かした高度な使われ方がなされている。

(3) 材料設計用科学技術データベース

超伝導材料を知識工学的手法を用いて設計するのを目的として開発中の科学技術データベース。910件の文献データと数100枚の図面[3)]がデータベースに入る予定になっている。図形配列，図形入出力，FIG型データベースの高度な使用例となろう。

(4) ロボットマニピュレータ用言語[4)]

マニピュレータのマクロな動作を関数として定義し，Petl システムに埋め込んだマニピュレータ用言語。回線を介してマニピュレータ制御用計算機と交信することができ，簡単にマニピュレータが動かせる。

(5) ロボット用環境モデルの作成[5)]

スーパインポーズディスプレイを用い，会話しながらロボットの実環境をリスト構造として Petl システムの内部に作成し編集するシステム。

(6) 分散計算機用プログラミングシステム

現在開発中の分散システム記述言語とそのプログラミングシステム。データベース，とバイナリ配列がGPIBを介し外部の分散計算機シミュレータと結合される予定になっている。

6. まとめ

本システムは数年前にその原型が開発され，その後改良につぐ改良が加えられたものである。COSMO UTS/VS 上で稼動中であり，利用者の記憶領域は3.5Mワードまでとることができる。

移植性を損わない範囲で，ホストオペレイティングシステムを十分に活用しまた協調するように配慮されている。現在，VAX/VMS に移植中であり，ACOS への移植も検討中である。説明書（基本システム編）も印刷中である。

今後の課題としては，データベースの同時更新機能の付加，強力なエディタの開発，説明書（プログラミングシステム編）の発行が挙げられる。

謝辞　本研究の機会を与えられた代々の部長，室長諸氏ならびに宇都宮大学助教授 白川洋充博士に謝意を表する。

1) Hirai, S.: Mechanical Troubleshooting System, 8th IFAC Word Congress, 1981
2) Ikeda, T: J-Analyser, IPSJ 3, 1981
3) 電子協：III-V族半導体レーザ用結晶データブック，S56
4) 末広他：構造的記述に基くロボット言語，S57 電気学会全国大会
5) 長谷川：ロボットの作業環境の教示，人工知能と対話技法研(81-4)
6) Tsukamoto, M.: Language Structures and Management Method in a Distributed Real-Time Environment, 3rd IFAC Workshop on DCCS, 1981