LispをPascalに翻訳し実行する，移植性の高い記号処理系

梅村 恭司
（日本電信電話公社 武蔵野電気通信研究所）

移植性の高い記号処理系を作製する目的で，ＬＩＳＰをＰＡＳＣＡＬに翻訳して実行できる処理系を開発した．この処理系は高い移植性と十分な性能を両立している．

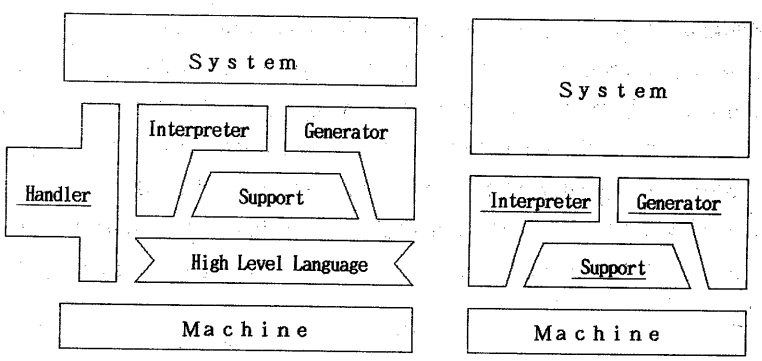
［1．Introduction］

移植性を考慮した処理系で，ＰＡＳＣＡＬなどは，その成功例である．ＰＡＳＣＡＬは中間コードを設定し，中間コードインタプリタを介してＰＡＳＣＡＬ－Ｐを動作させる．つぎに，そのＰＡＳＣＡＬ－Ｐを用いて真のマシンコードを生成するＴＲＵＮＣ－ＰＡＳＣＡＬを動作させる．さらに，この中間コードのＴＲＵＮＣ－ＰＡＳＣＡＬを用いて，マシンコードのＴＲＵＮＣ－ＰＡＳＣＡＬコンパイラを生成し，これを使用する．ＰＡＳＣＡＬに於ては，このような移植手法が成功を収めた．

処理形態がバッチ型であり，実行時サポートが単純であるＰＡＳＣＡＬ言語の場合には上記の方法で十分である．しかし会話型の処理系であり，複雑な実行時サポートが不可欠の処理系であるＬＩＳＰなどの記号処理言語の場合に，上記の方法では不十分である．ＰＡＳＣＡＬの処理系では，コンパイラそのものが重要な位置を占めているのに対し，記号処理系では，記憶域の管理，情報の入出力，などが重要な位置を占める．これらが上手に移植できないと処理系の移植性が低下する．

実行時サポートの全部を高級言語で記述し，ユーザのプログラムを高級言語に翻訳し，これら二つを組合わせて実行するという方針をたてた．fig(1)にＰＡＳＣＡＬとの比較図を示す．移植時に実行時サポートを機械語で用意しなくても，システムは動作する．マシンコードにアクセスしなくても，ユーザプログラムがマシンコードに翻訳されて実行できる．高級言語で処理系を記述するときに不足する機能があるが，高級言語へのコンパイラを動作させるときには，こういう機能を使用しないように注意した．同時に，このシステムで機種に依存する部分は極めて少量である．

システムの構造



今回の処理系の場合　　PASCALの場合

(注) 下線部を計算機ごとに用意する。

FIG. (1) PASCALとの比較

［2．Ｉｍｐｌｅｍｅｎｔ　ｌａｎｇｕａｇｅ］

インプレメント言語として，(1) ＰＡＳＣＡＬ，(2) Ｃ，(3) ＦＯＲＴＲＡＮ７７を考慮したが，ＰＡＳＣＡＬを採用した．ＣとＰＡＳＣＡＬを比較すると，最終的な性能はＣが良いけれども，Ｃのポインタ操作の言語仕様に不明確なところが多く，移植性を求めたときに障害がある．ＦＯＲＴＲＡＮとＰＡＳＣＡＬを比較すると，ＦＯＲＴＲＡＮに手続きの再起呼び出しと大域脱出との機能がないため，ＬＩＳＰからＦＯＲＴＲＡＮへの翻訳が不可能である．このため，ＰＡＳＣＡＬを採用することにした．

標準ＰＡＳＣＡＬで不足する機能がある．これは，おもに会話型のプログラム作成のときに必要となるものである．具体的には，(1) ユーザ割り込みの処理，(2) オーバーフローなどの処理，(3) ファイル名によるファイルアクセスがある．無秩序に標準から外れた機能をとりこむと移植性をそこなう．標準外の機能は，上の三点に限定した．

バッチ形態で動作可能なプログラムは，標準ＰＡＳＣＡＬの範囲で実行可能なように，標準ＰＡＳＣＡＬだけで動作するＬＩＳＰのサブセットを厳格に定めた．システムが用意しているＬＩＳＰ関数は上記の３点の関連のものを除いて標準ＰＡＳＣＡＬのだけで動作する．さらにＬＩＳＰからＰＡＳＣＡＬへのコンパイラについては，この標準ＰＡＳＣＡＬの範囲で動作することを保証した．

［3．Ｐｒｏｂｌｅｍｓ］

ＬＩＳＰの１関数をＰＡＳＣＡＬの１手続きに翻訳するという基本方針をたてた．そして，生成されたＰＡＳＣＡＬの手続きが，標準ＰＡＳＣＡＬの範囲であることを保証することにした．そして，これは実現できた．

このときに，いくつか問題が起きた．(1) 大域脱出の制御機構を実現すること，(2) 不用になった記憶領域を回収すること．(3) リスト形式の関数からＰＡＳＣＡＬ形式の関数へのコールと，ＰＡＳＣＡＬ形式の関数からリスト形式の関数へのコールとを両立させること．(4) 生成されたＰＡＳＣＡＬ手続きを実行時サポートルーチンと結合させること．この４点である．

この問題は，機械語に翻訳するときには起きない．(1) (2) (3) の問題点はＰＡＳＣＡＬの制御スタックをＰＡＳＣＡＬから直接には操作できないことに起因している．(4) の問題点は，ＰＡＳＣＡＬが実行中にコード部分を変更不可能であることに起因している．機械語では実現可能であるけれども，機械語以外の方法で上記のことを直接的に実現するのは困難である．

(1) (2) (3) は適切なガイドラインをＰＡＳＣＡＬコーディングに設定し，かつ，実行時サポートを注意深く作ることで解決できた．別のスタックを用意したり，間接的にＰＡＳＣＡＬのスタックを操作したりするのである．詳細の説明にはＬＩＳＰインプレメント固有の細かい知識を仮定せねばならず，細部に立入らなければならないと思われる．現実には，機械語を使用しなくても，標準ＰＡＳＣＡＬだけで解決できた．(4) は特別の機械語ルーチンを用意しないと原理的に不可能であるが，中間コードインタプリタとＰＡＳＣＡＬコードジェネレータとを用意して，疑似的に実現した．詳細は後述する．

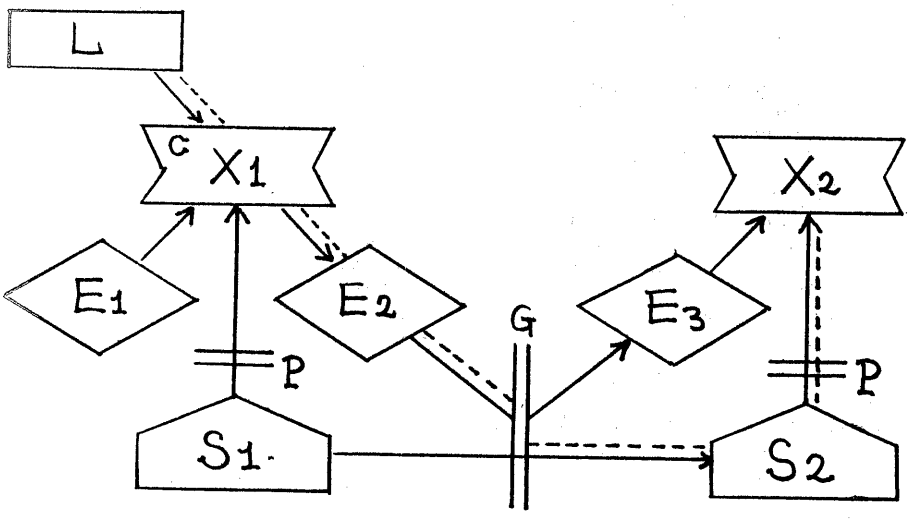
[4. Code generation]

今回のアプローチでは，機械語を生成する能力があるのは，ＰＡＳＣＡＬコンパイラだけと仮定したので，ユーザプログラムをコンパイルするのには，一旦，ユーザプログラムの翻訳結果を含むＰＡＳＣＡＬソースを，生成する必要がある．しかし，ＰＡＳＣＡＬソースを，実行時ルーチンと組み合わせて再コンパイルするのは，時間がかかる．これの回数は，へらしたい．そこで，ＬＩＳＰの関数は中間コードに逐次翻訳して，中間コードレベルでデバックし，最後に中間コードをファイルに書き出し，これを一斉にＰＡＳＣＡＬに翻訳することにした．

特に，中間コードでの実行結果と，ＰＡＳＣＡＬに翻訳したものでの実行結果は，完全に同一であるように注意した．これによって，ＰＡＳＣＡＬコードに翻訳してから機械語にコンパイルすることは，応用プログラムの最後に一回だけ行なえば良いので，全体のプログラム開発においてあまり負担にならない．

以下にＰＡＳＣＡＬへのコンパイル手順を記述する．環境には，中間コード，ＬＩＳＰ定数などが含まれる．はじめに，システムソース：Ｓ１と，初期環境：Ｅ１がある．Ｓ１がＰＡＳＣＡＬコンパイラ：Ｐで，Ｘ１という機械語になる．Ｅ１とＸ１で，ＬＩＳＰ処理系が動作する．

ユーザのプログラムは，fig (2) の破線の経路で機械語に変換される．ユーザのＬＩＳＰプログラム：Ｌは，ＬＩＳＰコンパイラ：Ｃによって中間コードに翻訳され，実行される．この時点で，Ｌを完全にデバッグする．そしてＸ１は，その中間コードを含んだ環境：Ｅ２を出力する．ＰＡＳＣＡＬコードジェネレータ：Ｇは，Ｅ２とシステムソースＳ１から，新しい環境Ｅ３と新しいシステムソースＳ２を生成する．Ｓ２は，ユーザプログラムのＰＡＳＣＡＬへの翻訳結果を含んでおり，Ｅ２の中間コードへのコール命令が，Ｅ３ではＰＡＳＣＡＬコードへのコール命令に変更される．このＳ２をＰＡＳＣＡＬコンパイラでＸ２という機械語に変換し，Ｘ２とＥ３でユーザプログラムが機械語で実行されるＬＩＳＰシステムが動作する．



ＦＩＧ．（2）コンパイルの流れ

約２千行のＬＩＳＰプログラムが１万４千行のＰＡＳＣＡＬのプログラムになり，現在のシステムソースは約２万行である．ＶＡＸ７８０－ＶＭＳで，この２万行のプログラムのコンパイルに２０分ほどかかる．ＶＡＸ－ＰＡＳＣＡＬの分割コンパイル能力を使用すれば，全体の再コンパイルの必要がないかもしれないが，移植性のためにこれは実現しなかった．

[５．Ｃｏｍｐｉｌｉｎｇ　ｅｘａｍｐｌｅ]

以下にＬＩＳＰからＰＡＳＣＡＬへの翻訳の例を示す．３つのＬＩＳＰ関数を３つのＰＡＳＣＡＬ手続に翻訳している．proc1,proc2,fib がfx6,fx7,fx8 に対応している．引数は，rvとr0というＰＡＳＣＡＬの変数を介して渡されている．ＬＩＳＰの定数は，varsという配列に格納される．この内容は，環境ファイルから読み込まれて初期化される．vars[0] には名前[n] が，vars[1] には名前[print] が，vars[2] には整数[0] が，vars[3] には整数[1] が，vars[4] には名前[+] が，その初期値としてはいる．print のコールに対応するものが，execfn(vars[1]) である．名前を介して呼ばれているのは，print の定義がリスト形式であり，コンパイルされていないことに起因している．コンパイルされている関数は，proc1,proc2 のように直接の手続呼び出しになる．

```
$ type example.lsp
(de proc1 () (proc1) (proc2) (print (cons n n)) ]
(de proc2 () (and (proc2) (proc1)) ]
(de fib (n)
  (selectn* n
     (0 0)
     (1 1)
     (() (+ (fib (sub1 n)) (fib (sub1 (sub1 n))))) ]
$ type example.pas
{ VARS } 10000;
{ CODE } 10000;
{ HEAD }
         6:fx6;
         7:fx7;
         8:fx8;
{ BODY }
procedure fx6; forward;
procedure fx7; forward;
procedure fx8; forward;
procedure fx6; begin
  fx6;
  fx7;
  rv.tag:=linktag[vars[          0].val];
  rv.val:=linkval[vars[          0].val];
  r0.tag:=linktag[vars[          0].val];
  r0.val:=linkval[vars[          0].val];
  begin consfn; end;
  execfn(vars[1]);
end;
procedure fx7; begin
  fx7;
  if rv.tag<>'n' then begin
    fx6;
  end;
end;
```

```
procedure fx8; begin
  stacks[stacker].tag:='w';
  stacker:=stacker+1;
  if stacker>stacklimit then trap('fn:stack');
  temp.val:=vars[          0].val;
  stacks[stacker].tag:=linktag[temp.val];
  stacks[stacker].val:=linkval[temp.val];
  stacks[stacker+1].tag:='b';
  stacks[stacker+1].val:=temp.val;
  stacker:=stacker+2;
  linktag[vars[          0].val]:=rv.tag;
  linkval[vars[          0].val]:=rv.val;
  rv.tag:=linktag[vars[          0].val];
  rv.val:=linkval[vars[          0].val];
  if rv.tag<>'i' then traptype;
  if (rv.val<0)or(rv.val>=          2) then
     rv.val:=          2;
  case rv.val of
    0: begin
      rv:=vars[2];
    end;
    1: begin
      rv:=vars[3];
    end;
    2: begin
      rv.tag:=linktag[vars[          0].val];
      rv.val:=linkval[vars[          0].val];
      if rv.tag='i' then rv.val:=rv.val-1 else traptype;
      fx8;
      stacks[stacker]:=rv;
      stacker:=stacker+1;
      rv.tag:=linktag[vars[          0].val];
      rv.val:=linkval[vars[          0].val];
      if rv.tag='i' then rv.val:=rv.val-1 else traptype;
      if rv.tag='i' then rv.val:=rv.val-1 else traptype;
      fx8;
      r0:=rv;
      stacker:=stacker-1;
      rv:=stacks[stacker];
      execfn(vars[4]);
    end;
  end;
  discard('w');
end;
$ logout
  UMEMURA       logged out at 14-AUG-1984 09:02:20.82M
```

fib の翻訳は宣言を何も行なわない場合で，安全であるが非能率なコードである．宣言によって，スタックの検査を外したり，変数の束縛を簡素化することができる．fx8 の最初の１１行が，スタック検査と変数束縛との処理である．変数[n] を大域変数としているので，このような複雑な処理となっている．

selectnはという制御構造は，整数の値による分岐を行なう．ＭＡＣＬＩＳＰのselectという制御構造に似ているが，分岐のもととなる値を整数に限る点で異なる．ＰＡＳＣＡＬに翻訳するときに，case文となる．最後の行の () は，上にある０と１以外の場合ということを意味している．ＰＡＳＣＡＬのコードでは，ラベル２が () に対応している．もしselectという制御構造を使用したら，if文が生成される．

+のコールに対応するのはexecfn(vars[4]) である．第一引数がrvに，第二引数がr0に，結果がrvによって渡されるという規約になっている．procl のconsと同じ形式であるが，第二引数の評価中に第一引数が保存されるようにスタック操作が必要である．第二引数が単純な形式であれば，スタック操作は行なわれない．

［6．Ｔａｒｇｅｔ　ｍａｃｈｉｎｅ］

このシステムは，スーパーミニコンピュータ以上の計算機で動作するように設計した．具体的な性能値は，１語３２ビット，メモリ空間１メガバイト，補助記憶２０メガバイト，そしてＰＡＳＣＡＬ言語が動作する計算機である．

このシステムは既に，(1) ＶＡＸ－ＶＭＳ，(2) ＶＡＸ－ＵＮＩＸ，(3) ＤＥＣ２０－ＴＯＰＳ，(4) ＭＥＬＣＯＭ－ＵＴＳ，で動作する．(1) (2) (3) は著者が行なった．(4) は電通大計算機学科の武市氏と天海氏と著者で行ない，ＵＥＤ－ＰＡＳＣＡＬのもとで移植が行なわれた．移植性は，これらの計算機の上で証明されている．

〔関数の定義〕

```
(de tarai (x y z)
    (cond ((> x y)
           (tarai (sub1 x) y z)
                  (tarai (sub1 y) z x)
                  (tarai (sub1 z) x y) ))
          (t y) ))
```

〔問題〕

```
(tarai 8 4 0)
```

〔実行時間〕

(1) NIL	list interpreter	170.2	sec.
(2) This LISP	list interpreter	147.6	sec.
(3) This LISP	code interpreter	22.2	sec.
(4) FRANZLISP	list interpreter	10.1	sec.
(5) This LISP	compiler	8.3	sec.
(6) FRANZLISP	compiler	6.3	sec.
(7) NIL	compiler	1.5	sec.

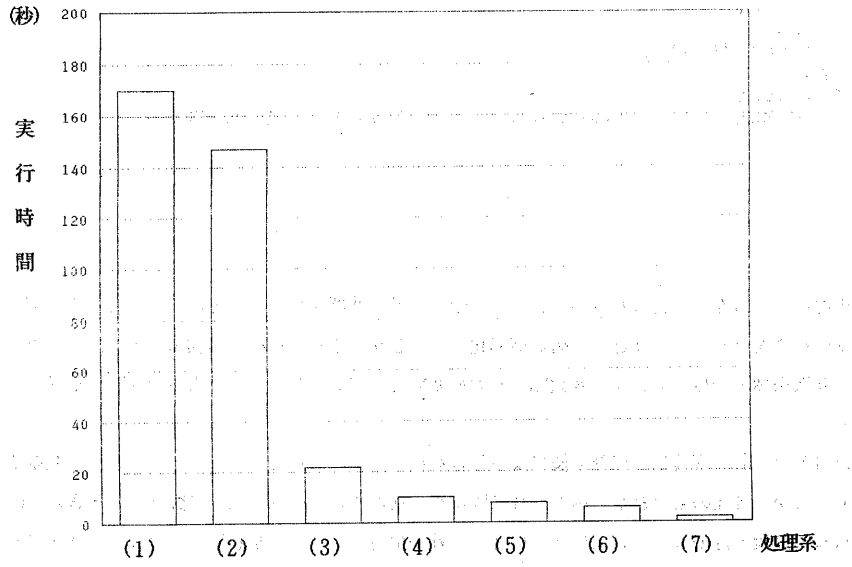


FIG. (3) 性能評価

［7．Ｅｆｆｉｃｉｅｎｃｙ］

セル空間は十分に広い．このシステムは，１語をそのままポインタアドレスに使用している．オペレーティングシステムが十分な空間を供給するなら，３２ビッド語のときギガセル級のＬＩＳＰである．実際にはオペレーティングシステムの制限によって使用できるセル数が決定される．

１セルは２リンクからなり，１リンクはタグ８ビットとアドレス３２ビットの合計４０ビットからなる．したがって，１セルが８０ビットとなる．ＵＴＩＬＩＳＰは１セル６４ビット，ＭＡＣＬＩＳＰ／ＤＥＣ２０は３６ビットである．アドレス部の大きさは，使用できる空間とつりあいをとる必要がある．最近の計算システムの性能からいって，１セル８０ビットは適当な値であろう．

処理速度については，ＶＡＸ－ＶＭＳのＰＡＳＣＡＬを使用した場合，ＶＡＸ－ＵＮＩＸ上のＦＲＡＮＺ－ＬＩＳＰと同程度の性能がでる．ＶＡＸ－ＵＮＩＸのＰＡＳＣＡＬを使用した場合，１０％ほど速度が低下する．速度は，その計算機のＰＡＳＣＡＬコンパイラの出来に左右される．とくに，手続き呼び出しと，定数添字の配列へのアクセスが実行速度を左右する．Fig(3)に，簡単な性能比較を示す．

［8．Ｐｏｒｔａｂｉｌｉｔｙ］

移植性という立場でみて，ＦＲＡＮＺ－ＬＩＳＰと比較する．ＦＲＡＮＺ－ＬＩＳＰは大部分がＣで書かれており，一部分が機械語である．機械語の部分なしでは動作しない．機械語の部分の移植に際しては，Ｃコンパイラのコード生成，ＬＩＳＰのインプレメント技術，オペレーティングシステムコールなどの広い範囲の知識を必要とする．

ＶＡＸ／ＶＭＳ版のこのシステムは，大部分を標準ＰＡＳＣＡＬで記述し，一部を拡張ＰＡＳＣＡＬで記述してある．機械語の記述は存在しない．拡張部分は，(1) ファイルのオープンとクローズ，(2) オーバフローなどのトラップ，(3) 割りこみ処理である．これらは，システムコールの知識を必要とするのみで，ＰＡＳＣＡＬのコード生成とかＬＩＳＰのインプレメント技法とかに対する知識は必要としない．

(1) (2) (3) は，本格的な応用プログラムを開発するときに常に必要となるものであるから，これを実現するための知識を持つ人材が多い．それに比較して，Ｃのコード生成とＬＩＳＰのインプレメント技法を同時に知る人は少ない．移植時に書きかえる量もさることながら，書きかえに必要な知識の量からも，ＦＲＡＮＺ－ＬＩＳＰより移植性の高いシステムである．

［9．Ｌａｎｇｕａｇｅ　ｓｐｅｃｉｆｉｃａｔｉｏｎ］

言語仕様はＭＡＣＬＩＳＰ系で，とくにＵＴＩＬＩＳＰに強く影響されている．入出力部分を除けばＵＴＩＬＩＳＰの拡張版である．１人年で作りあげたシステムであり，小型のＬＩＳＰである．しかし，実用プログラムを開発できるシステムである．

ＵＴＩＬＩＳＰからの拡張点は，(1) 大域脱出のときの強制評価(Unwind protection) (2) アドレス型のデータタイプ(General reference) (3) ベクトル型へのマップ関数の強化などがある．

[10. Conclusion]

ＬＩＳＰからＰＡＳＣＡＬへの翻訳が可能であることを実証して示した．高級言語で記述され，機械に依存する記述なしでもユーザプログラムが機械語で実行できるシステムが実現可能であることを実証して示した．高級言語へ翻訳するという移植性を重視した作り方をしたシステムが，実験システムにとどまらず，十分に実用に耐えることを示した．

[11. References]

(1) 近山　隆："ＵＴＩＬＩＳＰの開発，" 情報処理論文誌，Vol.24,No.5,pp599-604(1983)

(2) Chikayama,T."UtiLisp Manual,"Math. Eng. Tech. Rep. 81-6 Univ. of Tokyo(1983)

(3) 太田　義勝，吉田　雄二，稲垣　康善，福村　晃夫："Ｆｏｒｔｒａｎによって実現された会話型ＬＩＳＰシステムとその応用，" 情報処理論文誌，Vol.23,No.4,pp341-348(1982)

(4) Gris,M.L. and Hearn,A.C.:"A portable Lisp Compiler," Dept. of Computer Science, Univ.of Utah, Utah, (1979)

(5) Gris,M.L. and Hearn,A.C.:"A portable Lisp Compiler," Software, Vol.11,pp541-605(1981)

(6) Fitch, J.P. and Norman,A.C.:"Implementing Lisp in a High Level Language," Software, Vol.7, No.6 ,pp713-725(1977)