ＬＩＰｓ：　知識ベース開発システム

大久保　清貴
（パナファコム）

１．　はじめに

本文では，当社で開発中の知識ベース開発システムＬＩＰｓについて説明する。

知識ベースシステムは第五世代コンピュータプロジェクトの１つの課題にも成っているが，それは知識こそがこれからの貴重な資源となるからである[Moto-oka,1982]。現在アメリカを中心に医者，化学者，数学者などの知識を持ったシステムが実用化され始めている。

従来コンピュータは使用目的，実現方法が明確に解っている分野で使われてきたが，より広い分野で使われるためには，コンピュータに個人の知識を容易に与えられる手段が不可欠である。この手段を提供するのが知識ベース開発システムである。

知識ベース開発システムには二つの系統がある[Hayes-Roth,1983]。一番目は，まず知識ベースシステムが実際に作られてからアプリケーションに依存しない部分を抜きだしたもの（経験的アプローチ）で，ＭＹＣＩＮからＥＭＹＣＩＮが，ＰＲＯＳＰＥＣＴＯＲからＫＡＳが，ＨＥＡＲＳＡＹ－ＩＩからＨＥＡＲＳＡＹ－ＩＩＩとＡＧＥが作られた。二番目は，知識表現と推論機構を持った言語をベースとするもの（言語的アプローチ）で，ＯＰＳ５，ＥＸＰＥＲＴ，ＲＯＳＩＥ，ＲＬＬ，ＫＲＬ，ＦＲＬがある。

ＬＩＰｓ[Ohkubo, 1984]は，二番目の系統に属し，その主要なコンポーネントであるＬｉｓｐ，ＩＮＧＲＥＳ *)，Ｐｒｏｌｏｇと，複数形であることを示す’ｓ’により名ずけられた。

LIPs = (Lisp + INGRES + Prolog) * N

このシステムは，Ｐｒｏｌｏｇを主言語とし宣言的知識を表現するために，Ｌｉｓｐを手続き的知識を表現するために，ＩＮＧＲＥＳを画一的な大量のデータを扱うために備えている。またアルゴリズムの決っている計算や，高速の要求される箇所は，Ｃ言語を使うことができる。

ＰａｓｃａｌやＣに比べ数十倍も記述能力があるＰｒｏｌｏｇでプロトタイピングし，処理方式の固まった部分をＬｉｓｐやＣで高速化するといった方法でトップダウン開発が行える。また’ｓ’の示すように並行処理も可能である。

ＬＩＰｓの設計思想は，ユーザがシステムを最適な方法で構築できるような柔軟性のある統合された環境を提供することである。そのために，論理型プログラミング，関数型プログラミング，リレーショナル・データベース，オブジェクト指向プログラミング，快適なプログラミング環境といったソフトウェア技術の結集・融合を図った。

ＬＩＰｓの開発は，ＵＮＩＸ **)（パナファコム商品名ＵＮＩＣＵＳ）の上で約２年かけて行われ，現在アプリケーションを作ることによる有用性の検証と，Ｐｒｏｌｏｇ，Ｌｉｓｐより更にユーザに近いレイヤの設計中である。ＩＮＧＲＥＳの部分以外は新規に作ったのでスマートでコンパクトであるが非常にパワフルである。開発システムには，当社ミニコンピュータＰＦＵ－１５００をＴＳＳで用いていたが，十分なメモリ空間とＣＰＵ速度を得るためにパーソナル・ワークステーションへ展開中である。

－－－－－－－－－－－－

*) ＩＮＧＲＥＳは，Ｒｅｌａｔｉｏｎａｌ　Ｔｅｃｈｎｏｌｏｇｙ　Ｉｎｃ．の　登録商標　である。

**) ＵＮＩＸは，ＡＴ＆Ｔ　Ｂｅｌｌ　Ｌａｂｏｒａｔｏｒｉｅｓ　の商標である。

2. ＬＩＰｓのアーキテクチャ

図１にＬＩＰｓのアーキテクチャを示す。各コンポーネントの機能は次の通りである。

（１）　ＵＮＩＸ

流通ソフトウェアを取入れるためのパス。

（２）　Ｐｒｏｌｏｇ

宣言的知識を表現するのに用いる。ＤＥＣ－１０Ｐｒｏｌｏｇ[Clocksin,1981]と同じシンタックスである。高階述語（関係名が変数の述語や，アーギュメントが可変個の述語）が扱え，例外処理（端末からのブレークの処理など）もできる。

（３）　Ｌｉｓｐ

手続き的知識を表現するのに用いる。Ｌｉｓｐｋｉｔ[Henderson,1980]　Ｌｉｓｐに基づいていて，スマートな関数型言語である。Ｌｉｓｐで書いた関数は，読込む時にコンパイルされ，Ｐｒｏｌｏｇの述語として呼べるようになる。

（４）　ＩＮＧＲＥＳ

ＩＮＧＲＥＳは，カリフォルニア州立大学バークレー校で開発されたリレーショナル・データベースである。ＬＩＰｓからは大量の画一的データを管理するために用いる。ＩＮＧＲＥＳの機能をＰｒｏｌｏｇから呼べる述語が用意されている。

（５）　汎用ｐｌａｎｎｅｒ

アプリケーション向きの制限／手段記述をもとに，ゴールを達成する方法を導きだす。例えば，与えられた仕事を決められたコスト内で実現する方法を求めるのに使う。

（６）　汎用ｔｈｅｏｒｅｍ　ｐｒｏｖｅｒ

ユーザプログラムの性質を証明する。例えば，ある述語の２つのアーギュメントは同じメンバから成る，ということを証明する。

（７）　ｅｘｐｅｒｔ　ｓｈｅｌｌ

知識を獲得したり，論理の筋道を説明する。また，ユーザが簡単にメニュー方式プログラムやガイダンスを書けるｍｅｎｕｅ／ｈｅｌｐシステムを含む。更にＰｒｏｌｏｇをよりフレンドリな文法で使えるシンタックス・シュガーなどもこの機能にはいる。

現在，汎用ｐｌａｎｎｅｒ，汎用ｔｈｅｏｒｅｍ　ｐｒｏｖｅｒ，ｅｘｐｅｒｔ　ｓｈｅｌｌから成るレイヤを設計中である。このレイヤによって，プログラム開発システムから知識ベース開発システムへと発展する。

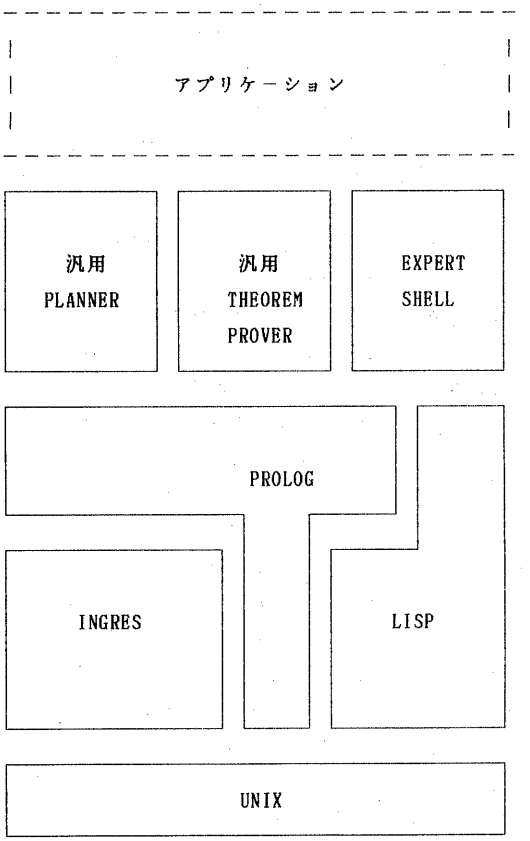

図１．　ＬＩＰｓのアーキテクチャ

3. ＬＩＰｓの特徴

ＬＩＰｓの特徴は　1．で述べたように，ソフトウェア技術の結集・融合を図った点である。本節ではその実現方法と意義を述べる。

3・1　マルチ言語

ＬＩＰｓはＰｒｏｌｏｇを主言語とし，Ｌｉｓｐ，ＩＮＧＲＥＳの機能が使える。

（1）　Ｌｉｓｐとの結合

Ｐｒｏｌｏｇのプログラムは，一階述語論理に制限を加えたホーン節の集まりとして定義される。２つのリストを結合するａｐｐｅｎｄは，

```
append([],X,X).
append([A|X],Y,[A|Z]):-
    append(X,Y,Z).
```

である。このプログラムはリストを２つに分けるために使うこともできる。これに対応するＬｉｓｐのプログラムは，

```
(append lambda (X Y)
    (if (null X) Y
    (cons (car X)
        (append (cdr X) Y))))
```

である。Ｌｉｓｐのプログラムの場合はリストの結合にしか使えないが，この用途でのみａｐｐｅｎｄを使う時は効率的である。またＬｉｓｐではａｐｐｅｎｄは組込み関数になっているので自分で定義する必要はない。

Ｌｉｓｐのａｐｐｅｎｄをプログの中から使うには，

```
append(X,Y,Z):=
    (lambda (X Y) (append X Y)).
```

とすればよい。ここで，頭はＰｒｏｌｏｇ，体はＬｉｓｐ，首は両者の混合になっている。このＬｉｓｐ部分は読込まれる時に自動的にコンパイルされる。同時に次のようなＰｒｏｌｏｇ流の文に変換される。

```
append(X,Y,Z):-
    lisp(append(X,Y,Z)),!.
```

ｌｉｓｐという述語は，アーギュメントＸ，Ｙが定数であることをチェックして，Ｌｉｓｐのａｐｐｅｎｄを呼ぶ。結果はＺにはいる。後ろの！は，バックトラックを禁止するためにある。

もう一つのＰｒｏｌｏｇとＬｉｓｐを結合するための述語として

```
prop(Name,Attribute,Value)
```

がある。これは，Ｌｉｓｐで便利に用いられるプロパティリストをＰｒｏｌｏｇからアクセスするもので，

```
prop(pen,color,V)
```

によりｐｅｎの属性ｃｏｌｏｒを変数Ｖに読んだり，

```
prop(pen,color,black)
```

で属性の値を書くことができる。

（2）　ＩＮＧＲＥＳとの結合

ＩＮＧＲＥＳのデータベース機能とインタフェースをとるために，次の述語を用意している。これは，Ｃ言語の中からＩＮＧＲＥＳの検索言語ＱＵＥＬを呼べるＥＱＵＥＬというプリプロセッサを使って実現された。

ａ．ＩＮＧＲＥＳを起動し，データベースを構成するリレーション名を求める。

```
ingres(Database,Relations)
```

ｂ．リレーションを構成する属性を求める。

```
relation(Relation,Attributes)
```

ｃ．リレーションのタップルを事実として読込む。

```
retrieve(Relation)
```

ｄ．リレーションを創成する。

```
create(Relation,Attributes)
```

ｅ．リレーションを壊す。

```
destroy(Relation)
```

ｆ．ＩＮＧＲＥＳを使い終わる。

```
quit_ingres(Database)
```

3・2　パラレル・ワールド

ＬＩＰｓの’ｓ’は，ＬＩＰという世界を多重に作

れることを表している。ＳＦの題材にされるパラレル・ワールドをモデル化したものであり，その目的は次の点である。

（１）　並列処理を行う。これにより，全系のスループットを高めると同時に，巨視的な横型探索（Ｐｒｏｌｏｇは縦型探索）を行うことができる。

（２）　システムを構築する時，全体をいくつかの独立性の強い専用モジュールに分割し，お互いは決められたプロトコルで連絡しあうという方法により，作成，管理，運用が容易になる。巨視的なオブジェクト指向プログラミングが可能となる。

次にパラレル・ワールドの実現方法を述べる。サブゴールＧ１，Ｇ２，Ｇ３のどれかが成功すればよいという条件は，

or(G1,G2,G3)

で表され，Ｇ１，Ｇ２，Ｇ３を順次実行し成功するか調べる。これと似た述語に，パラレル・ワールドを創成する

para(G1,G2,G3)

があり，サブゴールＧ１，Ｇ２，Ｇ３を並行に実行する。これは述語ｐａｒａで各サブゴールごとに子プロセスを作り，各々は別々のゴールを実行にかかることにより成される。同様に子プロセスもさらに子プロセスを作る事ができるので，結果的に家系図のような構造をつくりだすことができる（図２）。すべての子プロセスはＬＩＰ（Ｌｉｓｐ，ＩＮＧＲＥＳ，Ｐｒｏｌｏｇ）の機能をもつ。一番最初の親ＬＩＰだけは，ユーザとの間で会話処理を助けるためのモニタ機能（次節で詳述）をもつ。

このようにして作られた子プロセスが完全に独立し

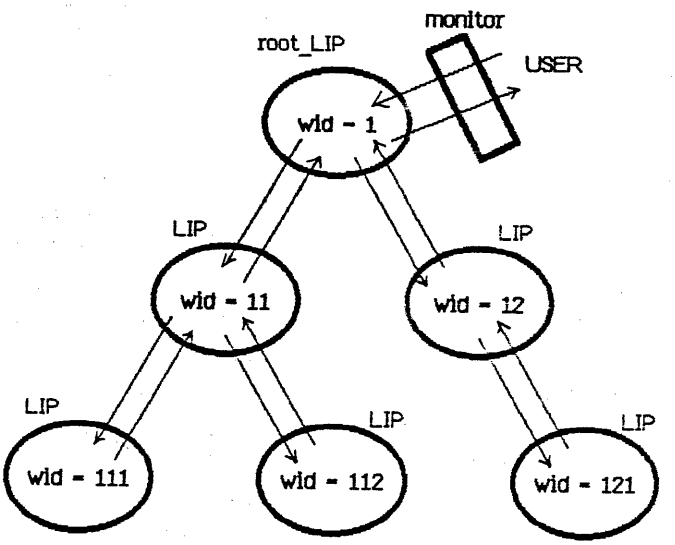

図２．　ＬＩＰｓの階層型パラレル・ワールド

て動くのでは用途が限られるので，直接の親プロセスとの間はＵＮＩＸのパイプという連絡路でつないでいる。ここを通して互いにメッセージを交換しあうことで，全体として協調した動きができる。メッセージの交換は通常のｒｅａｄ（Ｘ），ｗｒｉｔｅ（Ｘ）を拡張した

```
read(X,World_name)
write(X,World_name)
```

により行われる。Ｗｏｒｌｄ＿ｎａｍｅ（世界名）とは述語ｐａｒａのアーギュメントである。しかし，この名前は全系としてみた時に一意になる保証はないので，世界識別子というものをもうけた。これはｒｏｏｔ＿ＬＩＰ（一番最初の親）を１とし，その子供を１１，１２，・・とし，１１の子供を１１１，１１２，・・とする方法でつけられている。世界名も含めて世界識別子（Ｗｉｄ）と呼ぶ。Ｗｉｄにより，直接パイプでつながれていない世界の間でも親類を中継して交信する事ができる。また，ＬＩＰｓにはあるゴールを相手の世界で実行してもらう述語

```
request(Goal,Wid)
```

がある。これは，並行に走っているプロセス間でＰｒｏｌｏｇのユニフィケーションを行うものである。

３・３　プログラミング環境

知識ベース開発システムのように高度に知識集約的な場では，人的コストに見合う環境が必要である。ＬＩＰｓには，ＩＮＧＲＥＳの会話モニタからヒントを得て作った，ユーザとの会話を記録しておくエディタバッファを中心にしたコマンド群がある。

ユーザは通常ＬＩＰｓのプロンプト'＊'に対して，事実やルールの入力を行ったり検索などの実行を要求する。ここで構文の誤りがあった時，エディタコマンドの呼び出しにより簡単に修正できる。また事実やルールを部分的に変更して再実行するのも容易である。実行，再実行において構文エラーがあった時は，次のエディタ呼び出しでエラー部分がスクリーンの中心にくるようにしている。このコマンドは，ｖｉ（カリフォルニア州立大学バークレー校で開発されたフルスクリーン・エディタ）を呼び出すことで実現されている。

その他には，エディタバッファの内容をディスプレイに表示するコマンド，エディタバッファにファイルを読むコマンド，ファイルに書くコマンド，実行／再実行（Ｐｒｏｌｏｇのｃｏｎｓｕｌｔ／ｒｅｃｏｎｓｕｌｔ）コマンド，ＵＮＩＸコマンドインタプリタ（ｓｈｅｌｌ）を呼び出すコマンドなどがある。

次にデバッグ環境について述べる。Ｐｒｏｌｏｇでは，変数のバインディングがどのように行われ，具体値がいつ決まるのかは，たとえ自分の作ったプログラムでも解りにくい。これは，プログラムが宣言型であり，値を求める手順を記述したものではないからである。したがってデバッグ環境の良し悪しで生産性が数倍違ってくる。ＬＩＰｓには範囲トレース，指定述語トレースに加えブレーク・パッケージが用意されていて，１ステップ実行，ｗｈｙ型（なぜこのサブゴールを実行しようとしているか）およびｈｏｗ型（どのようにして前のサブゴールが成りたったか）の説明をさせることができる。Ｗｈｙ型の説明は，より幹に近い方のゴールツリーを表示することにより，ｈｏｗ型の説明は葉に近い方の既に達成されたゴールツリーを表示することにより成される。

良いプログラミング環境を作れる条件は，その言語のシンタックスの明瞭さと記述能力の大きさである。一つの基準として，ある言語のインタープリタを自分自身で書いた時のステップ数を提案する。簡単なものでＰｒｏｌｏｇは数ステップ，Ｌｉｓｐは数十ステップ，ＣやＰａｓｃａｌだと数千ステップであろう。ＰｒｏｌｏｇはＬｉｓｐに比べてプログラミング環境が悪いと言われているが，この指標でみると将来性がある。

４．　おわりに

ＬＩＰｓに似たシステムは，幾つか作られている。ＬｉｓｐとＰｒｏｌｏｇの融合は，ＬＯＧＬＩＳＰ[Robinson,1982]，ＱＬＯＧ[Komorowski,1982]，ＰＲＯＬ

ＯＧ／ＫＲ[Nakashima,1982]，ＱＵＴＥ[Sato,1983]に見られる。並列実行のＰｒｏｌｏｇは，Ｃｏｎｃｕｒｒｅｎｔ　Ｐｒｏｌｏｇ[Shapiro,1982]に見られる。オブジェクト指向／アクタモデルは，Ｓｍａｌｌｔａｌｋ[Goldberg,1983]，ＩＮＴＥＲＭＩＳＳＩＯＮ[Kahn,1982]に見られる。

ＬＩＰｓは，１．で述べたように，Ｐｒｏｌｏｇ／Ｌｉｓｐとユーザの間のレイヤを開発中である。現段階では商品化されていないが，ＵＮＩＣＵＳユーザに限りローカルな提供に応じている。ＶＡＸおよび６８０００ベースのＵＮＩＸシステムにも移植済みである。

ここ２～３年のＯＡ化の波は，安いマイクロプロセッサとメモリにより引き起された。次の波は，知識ベースを核としたシステムにより起ると考えている。

[文献]

[1] Clocksin, W. F. and Mellish, C. S.: "Programming in Prolog", Springer-Verlag (1981).

[2] Goldberg, A. and Robson, D.: "Smalltalk-80: The Language and Its Implementation", Addison-Wesley (1983).

[3] Hayes-Roth, F., Waterman, D. A. and Lenat, D. B.: "Building Expert Systems", Addison-Wesley (1983).

[4] Henderson, P.: "Functional Programming: Application and Implementation", Prentice-Hall (1980).

[5] Komorowski, H. J.: QLOG - The Programming Environment for Prolog in Lisp, "Logic Programming", Academc-Press (1982).

[6] Moto-oka, T., et al.: Challenge for Knowledge Information Prcessing Systems, "Fifth Generation Compuer Systems", pp.1-89, North Holland (1982).

[7] Nakashima, H.: Prolog/KR User's Manual, METR 82-4 University of Tokyo (1982).

[8] Ohkubo, K.: LIPs: Knowledge Base Development System, USENIX Summer Conference Proceedings, pp.151-158 (1984).

[9] Robinson, J. A. and Silbert, E.: LOGLISP: Motivation, Design and Implementation, "Logic Programming", Academic-Press (1982).

[10] Sato, M. and Sakurai, T.: QUTE: A Prolog/Lisp Type Language for Logic Programming, IJCAI-8, pp.507-513 (1983).

[11] Shapiro, E. Y.: A Subset of Concurrent Prolog and Its Interpreter, ICOT Technical Report, TR-003 (1983).