記号処理 20－3
（1982. 10. 18）

拡張PROLOG（ShapeUp）の実現について

横田 実* 梅村 護
（日本電気株式会社 C&Cシステム研究所）

1. はじめに

ShapeUpとは一階述語論理に基づくプログラミング言語PROLOGに文字列パターンマッチング機能をはじめとするいくつかの機能拡張を行った記号処理用プログラミング言語である。ShapeUpの処理系を開発したので概要を報告する。

計算機が身近に利用できるようになってくると従来の計算主体の処理から，文字や図形を扱う処理が中心になる。現状では図形処理の専用システムや文字列処理専用の言語（例えばSNOBOL）が利用可能であるが，それらは個々に独立して存在する。ShapeUpはこれらの文字，図形処理を，将来の応用を考えて統合化しようと試みたものであり，プログラムの基本制御構造として"述語論理"を採用し，その上で文字，図形を統一的に扱うことを目的としている。

以下，次節でShapeUpの特徴を簡単に述べ，3節では処理系の概要，4節では文字列処理の実現手法について述べ，最後にShapeUpの応用としていくつかのサンプルプログラムについて述べる。

2. ShapeUp[1]

ShapeUpは述語論理型言語PROLOG[2]を基本とし，文字列パターンマッチング機能の導入等の機能拡張を行ったもので，言語構文の基本はDEC-10版PROLOG[3]と同じである。プログラムの基本構成要素は「述語」であり「述語名」といくつかの「引数」により構成される。述語はいくつか組合わされてクローズ（節）を構成する。クローズは従来のプログラム言語におけるステートメントに相当するもので1つの完結した「関係」を表現している。クローズには以下の3種がある。

(i) アサーション　述語．　---｛1つの述語から成り，該述語が真であることを示す。[例] man(Tom).

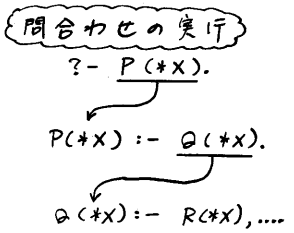
(ii) ルール
述語$_1$:- 述語$_2$, 述語$_3$, … 述語$_m$.　（AND結合）
述語$_1$:- 述語$_2$; 述語$_3$; … 述語$_m$.　（OR結合）
:-の右辺の述語がすべて（AND結合）もしくはいずれか1つ（OR結合）が真である時に限り述語$_1$は真であることを示す。[例] man(*X):- boy(*X).

(iii) 問合わせ　?- 述語$_1$, 述語$_2$, … 述語$_m$.
ルールの右辺のみの形式。右辺が真か否かの問合せ [例] ?- man(*who).

アサーション，ルールは事実，関係を宣言するもので一種のデータベースを構築する。これに対して問合わせが発せられると始めてプログラムの「実行」が開始され，問合せが真か否かを調べるためにデータベースの探索が行われる。具体的には真／偽を問う述語と同一の述語名を左辺に持つクローズ（アサーションまたはルール）を探し，対応する引数同士の一致を調べる。この時，一方が変数の場合には値の代入が行われる。2つの述語を同一化させるこの処理はユニフィケーショ

* 現在（財）新世代コンピュータ技術開発機構

ンと呼ばれる。 探索された述語がルールの場合には右辺が真であるかを調べるために，右辺の述語の呼び出しが更に続けられる。



述語の引数としては定数として文字列，数値，図形，構造体としてリスト構造，ストリング構造，及び任意構造体を記述することができる。また"*"を付加して変数を記述しても良い。

ShapeUpの特徴は①ユニフィケーションに文字列パターンマッチング機能を導入したこと ②データとして図形データも扱えるようにしたこと ③要素数可変の一次元配列としてのストリング構造を設けたこと④サイドイフェクトを利用するためのグローバル変数を導入したこと等である。

3. インタプリタ

プログラム言語の処理系にはコンパイラとインタプリタとがある。前者は処理効率の点で優れているが，プログラムの作成，デバッグといった開発の面からは後者のうが対話的であり柔軟性に富む。ShapeUpでは実行時に動的に決定される要因が多いためインタプリタ方式を採用した。但しプログラム入力時に決定可能な要素は極力，その時点で解決してしまう方針とした。例えばShapeUpの各クローズは意味的に独立しており，ほとんどの情報は1クローズ単位にコード化することが可能である。 インタプリタの開発にあたって特に留意した点は以下のとおりである。

(i) システムの修正，拡張の容易さ
(ii) プログラムもデータとして自由に扱えること
(iii) メモリ利用状況，スタック環境が明確に把握できること
(iv) 作成完了済プログラムの実行を最適化できるようにすること
(v) 述語の呼び出しはシステム外に対しても開かれていること（例えば他のプログラムの呼び出し）

インタプリタはポータビリティとインプリメントの容易さ，仕様変更の容易さからプログラム言語Cを用いて実現した。

3.1 システム構成

図1にシステム構成を示す。システムは入力したソースプログラムの内部形式を生成する前処理部と問い合わせの実行を行う実行部から成る。前処理部は 指定されたファイルもしくは端末装置からクローズを入力し，構文解析，文法エラーのチェックを行い中間結果であるテンプレート情報を出力する。これはクローズを構成する各構文要素（例えば変数名，引数情報等）を固定長の1アイテムに編集したテーブルで1クローズの入力毎に作られる。

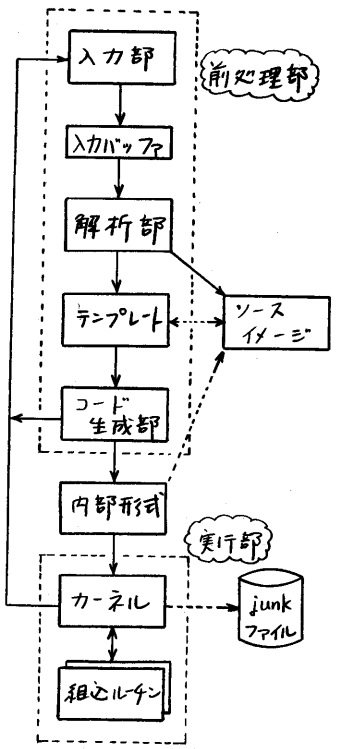


図1. インタプリタ システム構成

内部形式はこのテンプレートを走査しながら作成される。前処理部を構成する各モジュールは入力データからクローズを生成する場合（例えば組込述語 assert）などにも容易に利用できるよう考慮してある。

3．2　プログラムの内部形式

プログラムの内部表現形式は次の6種の構成要素をポインタで連結したものから構成される。（図2）

(i) プロセジャテーブル ------- 左辺に同一の述語名を持つクローズ群（プロセジャと呼ぶ）へのインデックスとして用いるテーブルで述語名，引数の数等の情報を持つ

(ii) クローズインデックス ------- 1クローズに対応する情報をまとめたもので，同一プロセジャに属する次のクローズへのアドレス，クローズの種類（アサーション，ルール等），クローズの左辺／右辺の情報へのアドレス等を持つ．

(iii) ヘッド引数テーブル ---------- クローズ左辺の述語に関するテーブルで各エントリは引数に対応して，引数データ・タイプ，タイプに応じたユニフィケーションルーチンエントリアドレスを持つ

(iv) ボディテーブル ---------- クローズ右辺の各述語毎に生成されるテーブルで呼び出す述語名，引数リストへのアドレス，次に実行すべき述語のボディテーブルアドレス等の情報を持つ．最適化のために呼び出すべき述語のクローズインデックスアドレス，あるいは組込述語の場合には実行サブルーチンのエントリアドレスを直接持てる。

(v) スケルトン ------- 構造体の表現のための構造体名，引数の数，引数リストのアドレスを持つ

(vi) リテラル ---------- 各引数のデータ表現でタイプ・タグと値から成る．

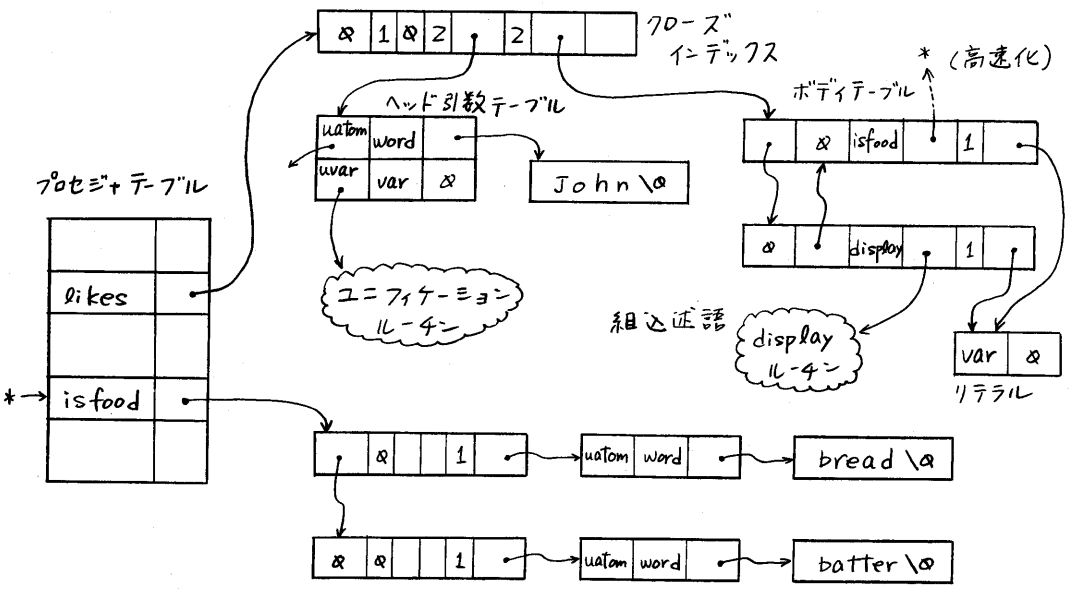


図2．　プログラムの内部形式

```
likes(John, *X) :- isfood(*X), display(*X).
isfood(bread).
isfood(batter).
```

3.3 実行メカニズムとスタック管理

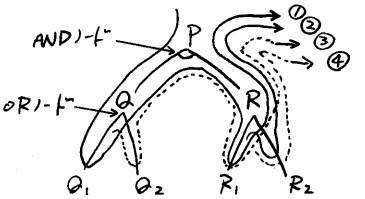
ShapeUp の基本実行メカニズムは PROLOG のそれと同一であり AND-OR トリーにより制御の流れを表現できる。 図3(イ)のプログラム例で，述語Pが真である可能性は①述語Q_1かつR_1が真，②Q_1かつR_2，③Q_2かつR_1，④Q_2かつR_2の4通りで図3(ロ)のAND-ORトリーにおける4本のパスに対応する。

```
P:- Q, R.
Q:- Q1.
Q:- Q2.
R:- R1.
R:- R2.
```

に対して

```
?- P.
```

の問合わせ



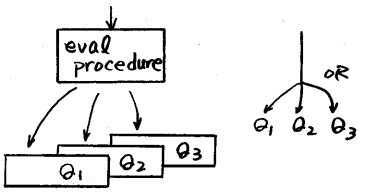

```
プロセジャ P
  CALL Q
  CALL R
  RETURN

プロセジャ Q
  CALL Q1 又は CALL Q2
  RETURN
```

図3(イ) プログラム例　(ロ) AND-ORトリー表現　(ハ) 手続き的表現

述語が真か偽かを判定する手順はプログラムの実行という点から図3(ハ)に示したように一種の関数呼び出しと考えることができる。(ハ)の表現はコンパイル済コードのイメージであり，インタプリタでは内部形式を解釈しながら述語呼び出し処理を行うことになる。即ち，ボディテーブルを順にたどりながら対応する述語の呼び出しを行う。 この時，ANDノードに対応する呼び出しはノードを構成するすべての述語の呼び出しが真となること。またORノードに対応する呼び出しはノードを構成する述語のいずれか1つが真であれば良い。 ShapeUpインタプリタのカーネル部はこの2種の呼び出しに対応した2つのルーチンが中心である。

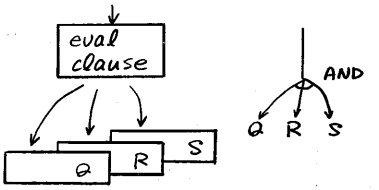
(1) evalprocedure (evalproc)

ORノードに対応する述語呼び出しを行うルーチンで述語呼び出しの最初のフェーズに相当する。(プロセジャの呼び出し) 入力情報として目的プロセジャに係る最初のクローズインデックスアドレスが渡される。クローズの評価そのものは(2)のevalclauseを呼び出すことにより行われるが，実行結果が偽であった場合にはリンクしている次のクローズ(インデックス)を実行する。真であるクローズが見つかり次第呼び出し元へリターンする。



(2) evalclause (evalcl)

ANDノードに対応する述語呼び出しを行うルーチンで1つのクローズについての評価を行う。入力情報として実行すべきクローズインデックスアドレスが渡される。 初めに当該クローズのためのスタックフレームを生成し実行環境の設定を行う。次に呼び出し側述語の引数と当該クローズ左辺の述語引数との間でユニフィケーションを行う。これには，クローズインデックスにリンクしているヘッド引数テーブルが用いられる。ユニフィケーションが成功し，かつ当該クローズがルールであれば更に，右辺の述語の呼び出しが行われる。このためにはボディテーブルの情報から(1) evalprocedureを呼び出す。従ってルーチンは2つであるが交互に呼び出されることになる。(図4)



evalprocedure，evalclauseは実行モードとして初めての呼び出しとバックトラック時の再実行の2通りある。また呼び出し元へのリターンコードとして次の4種

を返す。

(i) success　述語の評価が真
(ii) done　同上。他にバックトラックするものなし
(iii) fail　述語の評価が偽
(iv) error　述語の評価中にエラーを検出

クローズが変数を含んでいる場合，クローズの呼び出しが生じる毎に異なった値を持ち得るよう変数領域も動的に生成しなければならない。この実現は他の同種の言語処理系で行われているようにスタックを用いる。また呼び出し元へのリターンや評価が偽であった場合のバックトラックの管理のために制御用のスタックが必要である。DEC-10 PROLOGコンパイラ版[4]や，機械語で書かれたインタプリタでは連続領域にとったスタックとGOTO命令により効率の良い実行管理が行えるが関数型のサブルーチンとして実現するには若干の問題がある。一般には図5のように呼び出した各述語の変数領域とコントロール情報が共に1本のスタックの中に保存されている。バックトラックが生じるとあるポインタの指しているスタックフレームまでクリアされ対応する述語の呼び出し前の状況に復帰した後，他の可能性（ここでは述語R_2）が試みられる。その後スタック上に残されたコントロールリンクをたどりながら再実行が行われる。

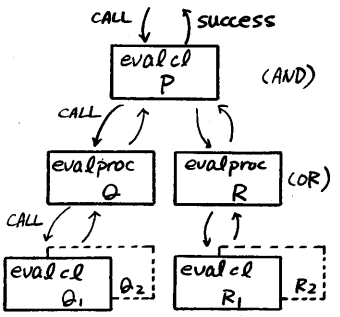


図4. evalprocedure および evalclause による実行

しかしevalprocedure／evalclauseのようなAND-ORノードに対応するルーチンを用いる場合には各ノード間がCALL／RETURNという対の制御構造で結ばれるため，上記のようなバックトラックを行おうとすると呼び出されないのにリターンだけ行うことになる。即ち図5においてスタックのクリアと述語R_2の呼び出しは実行できても述語R，Sへ戻ることができない。（CALLされていないため）従ってshapeUpでは図6に示したように処理効率は低下するが各ノードを順にたどり，戻るためのコントロールリンクを生成しながらバックトラック点へ到る方式を採っている。

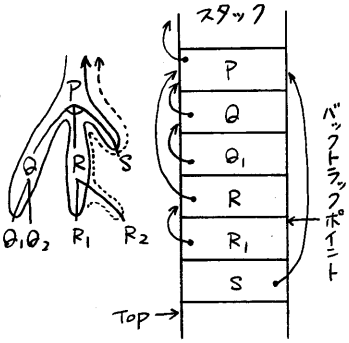


図5 スタック状況

この問題に対する他の1つの手法は述語の評価が完了してもRETURNせずに，そのまま下位ノードで呼び出し元（親）の仕事を引き継いで実行してしまう方式で，責任転嫁方式（Buck passing）と名づけた。この方式ではevalclauseにおいて処理が完了してもすぐにRETURNせず，呼び出し元が行うべき次の述語呼び出し処理をそのまま続けて実行してしまうもので実行履歴は図5と同じに完全に残る。（図7）いずれにしても通常のプログラム言語では制御の流れと環境の呼び出し／リターンが一致しているがPROLOGの実行では分離している，即ち制御は戻るがスタック（環境）は保存されるということに起因する。効率からはGOTO命令を用いる方が望ましいがプログラムは判りにくい。

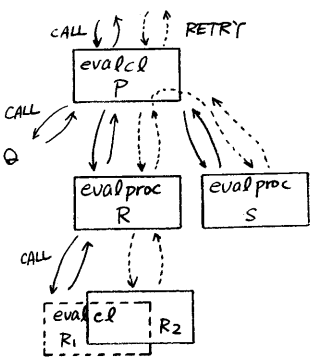


図6. バックトラック時のリトライ

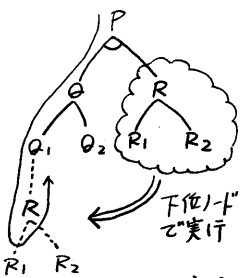
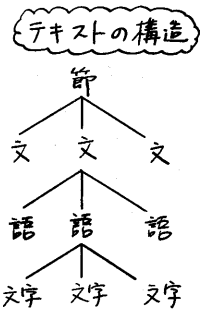


図7. Buck passing

4. 文字列処理の実現



ShapeUpでは文字列処理を単語レベル，テキスト・ストリングレベルの二段に分けて考えている。一般に扱おうとする文字列データは何らかの構造を持っているはずである。例えば通常の英文テキストは章，節，文，単語という構造を持っている。従って文字列処理といっても，一次元の文字の並びと考えるよりも含まれているデータ構造を陽に扱う方が高度の処理が可能になる。故に，基本データとしての物の名前，単語に相当する文字列とそれに対する文字の切り貼りを行う文字列処理と，基本データの配列としてのストリング構造とそれに対する各要素の切り貼りを行うストリング処理を設けた。

4.1 基本データレベルでの文字列処理

文字列パターンマッチングの機能としてはSNOBOL流[5]のマッチング機能から必要なものとして①文字列の連結（ShapeUpでの表示記号:^）②いくつかの文字列の選択（|）③任意の1文字との一致（井）④任意長，任意の文字列との一致（～）⑤1文字の繰り返し（m{許}）⑥マッチングした文字列の代入（変数{パターン}）を導入した。ShapeUpの特徴はこのパターンマッチング機能をユニフィケーションの中に含めたことである。

パターン・データは前処理部により，下図8に示した形式の内部形式に変換される。基本的には構造体であるが文字列マッチング処理がバイト単位になるため特殊コードを利用したバイトストリーム化を図った。連結(^)は連続領域にパックしたことにより実現しており対応するコードはない。パターンマッチングのユニフィケーションは次の手順で行われる。

P(*x^key^{a|bc}).

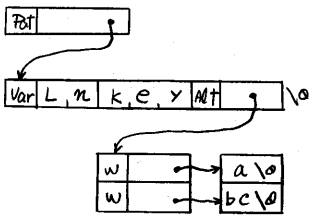


図8 パターンの表現

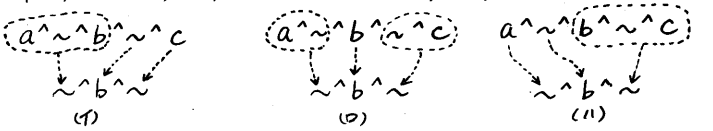
(i) 引数タイプの妥当性チェック
(ii) パターン・データから文字列を生成（一部のパターン用コードを含み得る）
(iii) 文字列の一致をチェック
(iv) 変数への値の代入（上記⑥の代入指定，または通常のユニフィケーションによるもの）

上記(ii)の処理にはパターン中に含まれている変数に対する値の読み出しも含まれる。また(iii)において不一致にな，たとしてもパターン中に文字列の選択が指定されていれば次の候補文字列を生成して再度パターンマッチを行う必要がある。更に(iv)においてマッチングした文字列の中に未定義変数が含まれている場合にはパターン構造を生成した後，代入を行う。

4.2 マッチング・マトリクス

パターンが任意長，任意の文字列と一致する記号(～)を含んでいる場合には幾通りかのマッチングの仕方が存在し得る。PROLOGの世界ではこのような時後続する述語呼び出しの失敗によりバックトラックするのが普通であろう。しかしそのような実現では①制御の複雑化，②バックトラックに備えて保存する情報量の増大の点から余り現実的ではない。従ってShapeUpではマッチング結果を一意に決める，しかもなるべく人間にとって自然になるように決めること，及びその制限で利用できる範囲の複雑度にパターンマッチングの利用法を留めるという方針を採った。例えばパターン"a^～^b^～^c"と"～^b^～"の一致は何通

りかの一致の仕方が存在するが，以下の(ロ)が最も自然なマッチングと言えるであろう。即ち，一致した文字（文字列）を着点としてパターンの一致を考える。



未定義変数は任意の文字との一致が可能であり"~"と同一の意味を持つ。即ち上記の例はP(a^*X^b^*Y^c)とP(*X^b^*Y)なる述語のユニフィケーションにおいて変数にどのような値をセットするかという問題と同値である。

	a	~	b	~	c
~	X	X	×	×	×
b	α	×	1	×	α
~	×	×	×	X	X

このような複雑なパターンマッチングを実行するためにShape-Upではマトリクスを用いた手法を採用している。原理は人間が文字列パターンを並列に眺めてマッチングを調べる動作を真似たものである。まずの対応する文字同士を比較し，②一致なら"1"，不一致なら"α"，任意長，任意の文字（もしくは未定義変数）との一致ならdon't care "X"をマトリクスにセットし，③マトリクスの左上隅より右下隅まで"1"もしくは"X"をたどって到達できれば（これも一致経路と呼ぶ）マッチング成功と見なす。この一致経路の形成の際に一致した文字位置"1"を重視することにより上述の自然なマッチングを達成できる。この方式は逐次型マシン上で実現する限り余り高速ではないが，将来のハードウェア化は容易であり高速化が期待できる。更にShape Upでは文字列処理を単語レベルのデータに留めているため比較的短い文字列の扱いが中心となりハードウェア化に向いている。

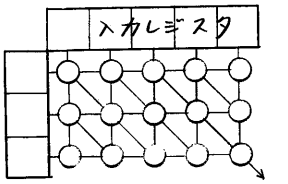


図9 マッチングマトリクスとハードウェア化

4.2 ストリングレベルでの文字列処理

ShapeUpでのストリング構造とは任意のデータ項目（構造体でも良い）を一次元配列として連結したもので，構成要素が文字列データであれば言わゆる文字列ストリングに相当する。図10にその内部表現を示す。このレベルでのマッチング機能としては①任意の1要素との一致(_)，②任意長，任意の構成要素との一致(~)を備えている。例えばP(<This△is△_>)は"This△is"で始まる3要素から成るストリング構造を引数に持つ述語Pとユニフィケーション可能であり，P(<This△is△~>)の場合なら3ストリング構造の3要素以降に何があっても構わない。基本データレベルでの文字列処理の場合と同様に未定義変数は任意のデータ構造と一致し，"~"と同値である。ユニフィケーションによってストリング構造のネストが生まれる場合によっては1レベルのストリング構造と見なす必要も生ずる。このためには図11に示すようにレベルを変えるストリング構造を備えている。ストリング構造の要素として任意のデータタイプが許されており，この結果，テキスト中でも自由に数値，図形データを挿入することができる。

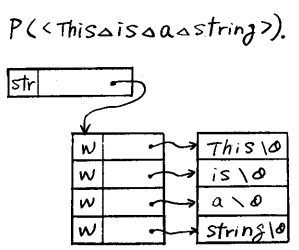


図10 ストリング構造の表現

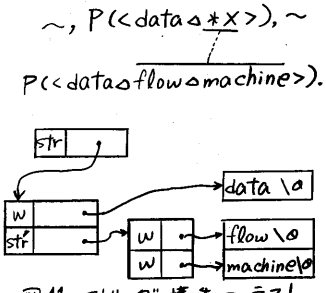


図11 ストリング構造のネスト

5. 応用例

ShapeUp の記述性を調べるためのいくつかのサンプルプログラム例を以下に示すが，いずれもプログラムは視覚的で判り易い。

(1) 文字列処理の例

英単語の語尾変化

```
past(*X^d, *X).
past(*X^ed, *X).
past(*X^ied, *X^y).
  ↓
過去形より原形を抽出する
```

(2) 曖昧検索の例

電話帳

```
Tel(system-kenkyu-bu, 1221).
Tel(computer-seizo-bu, 2112).
  ↓
に対し問合せ
  ?- Tel(*X^-seizo-bu, *Y).
により一部分の情報から索く
```

(3) テキスト処理の例

文の検索

```
find(<*topline△～>, *key, *topline):-
                check(*key, *topline).
find(<_△*remline>, *key, *line):-
        find(*remline, *key, *line).
check(*key, <～△*key△～>).
```

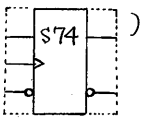
に対して問合せ

```
?- find(*text, keyword, *line).
```

によりテキスト中のキーワードを含む文を見つける

(4) 図形処理の例

図形に名前をつけて管理することも出来る

IC(74S74, 

).

6. おわりに

文字，図形を容易に扱うことを目的としたプログラミング言語 ShapeUp の処理系について述べた。システムは実験，評価を主目的としているため性能よりも作り易さを重視して作られている。現在，当研究所では ShapeUp 言語仕様の見直し，インタプリタの改良，最適化手法の導入を進めているところであるが，述語論理に基づくプログラミングスタイルと文字，図形が扱えるという ShapeUp の特徴は今後，高度なマンマシンインタフェースを構築するツールとして有効であると考えている。更に ShapeUp を効率良く実行できる高性能パーソナルマシンの設計も並行して進めている。

最後に本研究の機会を与えて下さった当研究所三上所長代理，箱崎部長，山本課長，インタプリタの開発に協力頂いた(株)日本タイムシェア小西氏，(株)日本情報研究センタ竹部氏，並びにCS研究部諸氏に深謝します。

参考文献

[1] 横田，梅村 "PROLOGの記号処理への機能拡張" 記号処理研究会資料 19-2，1982

[2] kowalski, R. "Logic for Problem Solving" North-Holland, 1979

[3] Pereira, L.M. et al. "User's Guide to DECsystem-10 PROLOG" DAIR, Univ. of Edinburgh, 1974

[4] Warren, D.H. et al "Implimenting of PROLOG: compiling predicate logic programs" DAIR, Univ. Edinburgh 1977

[5] Griswold, R.E. et al "The SNOBOL4 Programming Language" 2nd Edition, Prentice-Hall, 1971