プログラミング-言語・基礎・実践- 21－12
(1995. 3. 9)

Pascal から Hichart へのトランスレータの属性グラフ文法による記述と Prolog による実現

大井裕一
東洋大学工学部
350 埼玉県川越市
鯨井中野台2100
0492-31-1135
ohi@krc. toyo. ac. jp

安達由洋
東洋大学工学部
350 埼玉県川越市
鯨井中野台2100
0492-31-1135
adachi@krc. toyo. ac. jp

夜久竹夫
日本大学文理学部
156 東京都世田谷区
桜上水3-25-40
03-3329-1151
yaku@chs. nihon-u. ac. jp

あらまし

プログラムの図式表示は，プログラム情報の視覚化や視覚的プログラミングツールの実現のために不可欠である．Hichart は木フローチャートを基礎図式とする図式型言語であり，プログラムの制御の流れと階層構造を同時に表示できる．本研究では Pascalから Hichart への変換を属性グラフ文法で定式化し，文法に基づくトランスレータを Prolog で実現した．この属性グラフ文法は，ISO 規格に準拠した Pascal のプログラムに対して，文脈自由グラフ文法による構文規則で Hichart 図式を生成し，また，属性評価で配置されるセルの位置や大きさなどが計算される．なお，得られた属性をもとに，Hichart は OSF/Motifを用いて X Window 上に表示される．

キーワード　プログラムの図式表示，Hichart，属性グラフ文法，Prolog，構文解析，属性評価

Attribute Graph Grammar for Hichart-Pascal Translator and Its Inplementation in Prolog

Yuichi Ohi
Faculty of Engineering,
Toyo University
2100, Kujirai, Kawagoe-shi,
Saitama 350
0492-31-1135
ohi@krc. toyo. ac. jp

Yoshihiro Adachi
Faculty of Engineering,
Toyo University
2100, Kujirai, Kawagoe-shi,
Saitama 350
0492-31-1135
adachi@krc. toyo. ac. jp

Takeo Yaku
College of Humanities and
Science, Nihon University
3-25-40, Sakurajousui,
setagaya-ku, Tokyo 156
03-3329-1151
yaku@chs. nihon-u. ac. jp

Abstract

Program diagrams are essential to visualize program structures and to realize visual programming tools. Hichart is one of diagram description languages based on tree flowcharts, and can display control flows and hierarchical structures of programs simultaneously. We have formulated an attribute graph grammar to translate Pascal programs into Hichart-diagrams, and have developed a translator based on it in Prolog. The grammar generates Hichart-diagrams for Pascal based on ISO standard, with syntax rules in a context-free graph grammar, and computes attributes, such as cell's position, size, etc. With these attributes, a Hichart browser displays Hichart by using OSF/Motif widgets on X Window.

key words　program diagrams, Hichart, attribute graph grammar, Prolog, syntax analysis, attribute evaluation

Pascal から Hichart へのトランスレータの属性グラフ文法による記述と Prolog による実現

Attribute Graph Grammar for Hichart - Pascal Translator and Its Implementation in Prolog

○大井裕一† 安達由洋† 夜久竹夫††
Yuichi OHI Yoshihiro ADACHI Takeo YAKU
†東洋大学工学部 ††日本大学文理学部
Faculty of Engineering, Toyo University College of Humanities and Science, Nihon University

1．はじめに

プログラムの図式表示は，視覚的プログラミングツールの実現やソフトウェア事例データベースのプログラム情報の視覚化のために不可欠な機能である．例えば，プログラム開発の際にはプログラムの階層木構造がそのまま図式表示に反映されているので全体構造が即座に把握でき，また部分的な修正，変更，トレースも容易に行うことが可能となる．ソフトウェア事例データベースからプログラムを参照あるいは再利用する際にも，プログラムを図式表示することによりプログラムの視認性と解読性が向上し検索や利用を容易にする．

Hichart[1],[2]は，木フローチャートを基礎図式とする図式型言語である．プログラムの制御の流れと階層構造を同時に表示できるという特徴を持ち，構文指向の視覚的プログラミングやプログラム可視化ツール開発に適した言語である．

本研究では，Pascalプログラムから Hichart への変換を属性グラフ文法[3],[4]で定式化し，この文法に基づくトランスレータを Prolog で実現した．Hichart の属性グラフ文法による記述例は文献[3],[4]で既に報告されているが，本研究では，ISO 規格に準拠した Pascal[5]に対する完全な属性グラフ文法を定義して，この文法に基づくトランスレータを実現している．

また，本研究で Prolog を記述言語としたのは，文脈自由文法による構文解析の記述が容易であり，さらに変数のまま属性評価式を生成し必要なときに式の値を計算する部分計算が可能で効率の良い属性評価器が実現できるという理由による．

なお，変換された Hichart は X Window と OSF/Motif ウィジットを利用したビットマップスクリーン上に表示される．

2．トランスレータの属性グラフ文法による定式化

Pascal プログラムから Hichart への変換問題を属性グラフ文法で記述する．

属性は以下に示す継承属性と合成属性から成る．継承属性の x, y, j はセルの接続関係を求めるため，合成属性としても用いられる．

継承属性

- x, y ： 配置されるセルの座標
- RootX ： ルート（一番左の）セルの X 座標
- TopY ： 最も上に配置されるセルの Y 座標
- MinW, MinH ： セルの最小幅，最小の高さ
- GapX, GapY ： セルとセルの間隔
- j ： セルの識別番号

合成属性

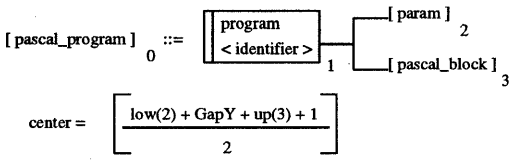
- h, w ： セルの幅，高さ（非終端記号はこれらの属性を持たない）
- up, low ： 指定されたセル，または非終端記号を根とする部分木に必要な空白で up は根より上の空白， low はセルの場合はセルの高さも含む）
- n, str ： セルの種類（番号），内部の文字列
- line ： セルとセルを結ぶ線
- ll ： セルラベル（セルの左上につく）
- cl ： 条件ラベル（セルの左肩につく）
- nc ： セルの数

なお，構文規則，属性評価中の記号，関数の意味は次の通りである．

- 開始記号 = [pascal_program]
- [x（計算式）] : x を超えない最大の整数
- "●" : 辺の形を整えるためのダミー接点
- < ... > : 文脈自由文法における非終端記号
- get_str(X) : セル内の文字列を求める関数
  (= [str_1, str_2, ..] ... 各 str_i はセル内の i 行目に入る文字列)
- get_width(X), get_height(X) : 文字列のリスト X からセルの幅、高さを求める関数
- get_line(S,E) : 始点セル S, 終点セル E を結ぶ線を求める関数

以下に，主な非終端記号の書き換え規則と属性評価式を示す．

① 開始記号



up(0) = up(2) + center　　low(0) = center + low(3)
x(1) = RootX　　y(1) = TopY + center
x(2) = RootX + MinW + GapX　y(2) = TopY
x(3) = RootX + MinW + GapX　y(3) = TopY + 2*center
j(1) = 0　j(2) = 1　j(3) = 1 + nc(2)
w(1) = MinW
h(1) = get_height(['program',<identifier>])
n(1) = 2
str(1) = get_str(['program',<identifier>])
line(1,2) = get_line(1,2)　　line(1,3) = get_line(1,3)

図2.1　開始記号の書き換え規則と属性評価式

- 終端記号１の X 座標と非終端記号２の Y 座標にはそれぞれ定数を割り当てる．
  （記号２は最も上に，記号１は最も左にそれぞれ配置される）
- Y 座標において終端記号１は非終端記号２，３の中間に配置される．
- 非終端記号０に必要な高さ(up(0), low(0)) は，書き換え規則の右辺にある記号に必要な高さから求める．

図2.1 の開始記号に関する属性は以下のように図示できる．

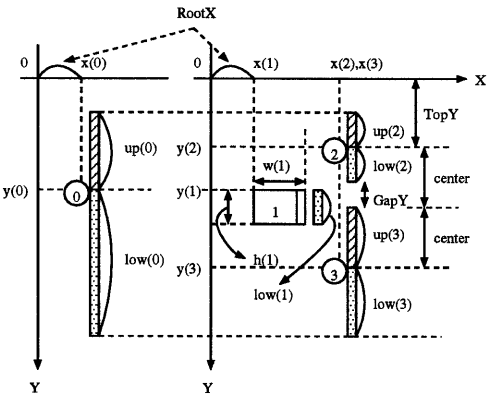
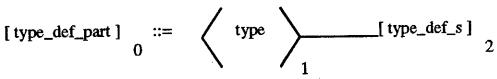


図2.2　開始記号の属性評価

② 型宣言部



up(0) = up(2)　　low(0) = low(2) + GapY
x(1) = x(0)　　y(1) = y(0)
x(2) = x(0) + w(1) + GapX　　y(2) = y(0)
nc(0) = 1 + nc(2)
j(1) = j(0)　j(2) = j(0) + 1
w(1) = MinW
h(1) = get_height(['type'])
n(1) = 15
str(1) = get_str(['type'])
line(1,2) = get_line(1,2)

図2.3　型宣言部の書き換え規則と属性評価式

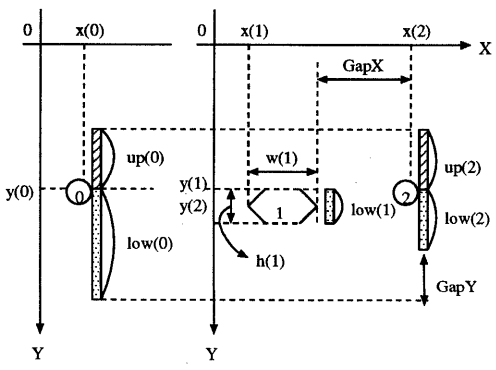


図2.4　型宣言部の属性評価

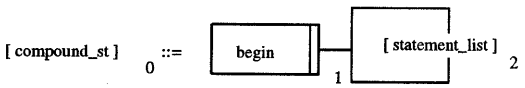
プログラムによっては，型宣言部が空の場合もある．このような場合の書き換え規則と属性評価は以下のようになる．

[type_def_part] ::= ●
0

up(0) = 0　　low(0) = 0
nc(0) = 0

図2.5　型宣言部が空の場合

③ 複文



up(0) = up(2)　　low(0) = low(2)
x(1) = x(0)　　y(1) = y(0)
x(2) = x(0) + w(1) + GapX　　y(2) = y(0)
nc(0) = 1 + nc(2)
j(1) = j(0)　j(2) = 1 + j(0)
w(1) = MinW
h(1) = get_height(['begin'])
n(1) = 10
str(1) = get_str(['begin'])
line(1,2) = get_line(1,2)

図2.6　複文の書き換え規則と属性評価式

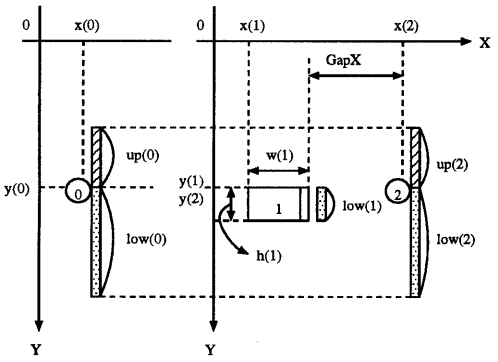


図2.7　複文の属性評価

上記の規則では非終端記号の上に縦線が引かれているが，実際は次のように描画される．

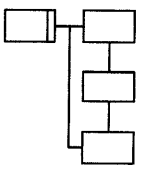
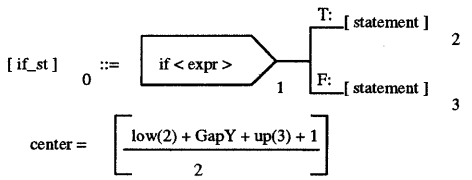


図2.8　begin セルの描画例

④ if then else 文



center = [(low(2) + GapY + up(3) + 1) / 2]

up(0) = up(2) + center　　low(0) = center + low(2)
x(1) = x(0)　　y(1) = y(0)
x(2) = x(0) + w(1) + GapX　　y(2) = y(0) - center
x(3) = x(0) + w(1) + GapX　　y(3) = y(0) + centerup(2)
nc(0) = 1 + nc(2) + nc(3)
j(1) = j(0)　j(2) = 1 + j(0)　j(3) = 1 + j(0) + nc(2)
w(1) = MinW
h(1) = get_height(['if',<expr>])
n(1) = 12
str(1) = get_str(['if',<expr>])
cl(2) = T　　cl(3) = F
line(1,2) = get_line(1,2)　　line(1,3) = get_line(1,3)

図2.9　if then else 文の書き換え規則と属性評価式

・Y 座標において終端記号 1 は非終端記号 2，3 の中間に配置される．
・if then 文の場合は，条件は真のみであるから，非終端記号 2 は終端記号 1 と横並びに配置される．

if then else 文の属性評価式は以下のように生成される．

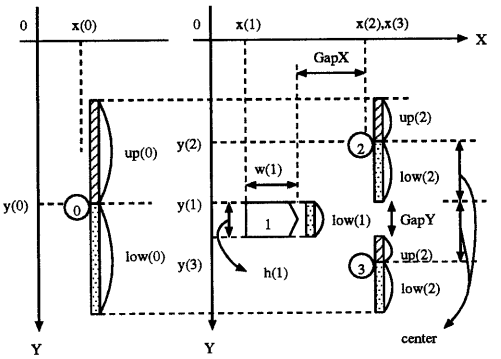


図2.10　if then else 文の属性評価

・条件ラベルは，非終端記号 2，3 の位置に導出されるセルの左肩にそれぞれ T（真），F（偽）が付けられる．
（if then 文の場合は，T（真）のみがセルの左肩に付けられる．）

⑤ 代入文

[assign_st]$_0$ ::= < variable > := < expr >$_1$

up(0) = 0
x(1) = x(0)
nc(0) = 1
j(1) = j(0)
w(1) = get_width([<variable>,' := ',<expr>])
h(1) = MinH
n(1) = 1
str(1) = get_str([<variable>,' := ',<expr>])

low(0) = h(1)
y(1) = y(0)

図2.11 代入文の書き換え規則と属性評価式

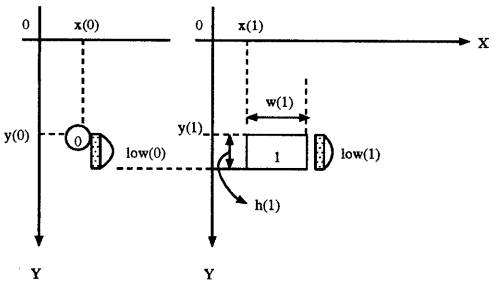


図2.12 代入文の属性評価

・代入文の書き換え規則の右辺は木構造の「葉」にあたるため，h は最小値をとり， w は文字列の長さから求める．

3．属性グラフ文法のProlog による実現

コンピュータ内では，図形データも記号表現する必要がある．そこで本トランスレータでは，Hichart の各セルデータを次のように Prolog の項として内部表現した．

```
hichart_code(id(File,J),cell(N),location(X,Y),
        size(W,H),[Parent,ChildList,Up,Low],
            [label(L),cond(C),string(STR)]).
```

また，セルとセルの接続線データは次のように表現した．

```
connect_code(id(File,J1),id(File,J2),Linedata).
```

ただし，Linedata は [[x1,y1],[x2,y2],..]のリスト構造であり，x_i，y_iは各々接続線が通る点の X 座標 Y 座標データである．

トランスレータは，Pascal から Hichart内部表現への変換部と，内部表現を元に X Window 上に表示する描画部とから成る．

以下に，Prolog による Hichart 内部表現への変換部の記述（一部）を示す．

（1）開始記号

```
/*******************************************************/
/*      pascal_program(main)                           */
/*******************************************************/
pascal_program(File,[`program`,I,`(`|T],P) :-
        identifier(I),
        concat([`program `,I],Atom1),
        assertz(pre_defined_id(I)),
        !,

/* ------------------------------- */
        hichart_cell_gapX(GapX),
        hichart_cell_gapY(GapY),
        hichart_cell_TopY(TopY),
        hichart_cell_RootX(RootX),
        N1 = 2,
        get_cell_size(not_leaf,N1,Atom1,W1,H1,Str1),
        Center = (Low2+GapY+Up3+1)//2,
        X1 = RootX,
        Y1 = TopY + Center,
        X2 = RootX + W1 + GapX,
        Y2 = TopY,
        X3 = RootX + W1 + GapX,
        Y3 = TopY + 2*Center,
        J1 = 0,
        J2 = 1,
        J3 = 1 + Nc2,
/* ------------------------------- */

        param(File,[`(`|T],[`;`|R],PPARAM,
                [_Up2,Low2,X2,Y2,J2,Nc2],ZP),
        pascal_block(File,R,[`.`],PBLOCK,
                [Up3,_Low3,X3,Y3,J3,Nc3],ZB),
        get_line([id(File,J1),location(X1,Y1),size(W1,H1)],
                [ZP,ZB],Connect1),
        PID = [id(File,J1),cell(N1),location(X1,Y1),size(W1,H1),
                connect(Connect1),cellList([string(Str1)])],
        append(PID,[PPARAM|PBLOCK],P).
```

図3.1 開始記号の解析

（2）if then else 文

```
/*******************************************************/
/*              if_st                                  */
/*******************************************************/
if_st(File,[`if`|T],Ts,P,[Up0,Low0,X0,Y0,J0,Nc0],Z) :-
        split_list(`then`,T,T1,T2),
        expression(T1,[],PEXPR),

        concat([`if `,PEXPR],Atom1),

        statement(File,T2,R,PT,[Up2,Low2,X2,Y2,J2,Nc2],ZT),

/* ------------------------------- */
        N1 = 12,
        get_cell_size(not_leaf,N1,Atom1,W1,H1,Str1),
        C12 = `T:`,
/* ------------------------------- */
        PT = [IDT,CELLT,LOCATIONT,SIZET,cellList(LISTT)|RPT],
        PT2 = [IDT,CELLT,LOCATIONT,SIZET,
            cellList([cond(C12)|LISTT])|RPT],

        /****************
           if then else
         ****************/
        R = [`else`|RELSE],
```

```
/* ------------------------------ */
        hichart_cell_gapX(GapX),
        hichart_cell_gapY(GapY),

        Center = (Low2+GapY+Up3+1)//2,
        Up0 = Up2+Center,
        Low0 = Center+Low3,
        X1 = X0,
        Y1 = Y0,
        X2 = X0+W1+GapX,
        Y2 = Y0-Center,
        X3 = X0+W1+GapX,
        Y3 = Y0+Center,
        Nc0 = 1+Nc2+Nc3,
        J1 = J0,
        J2 = J0+1,
        J3 = J0+1+Nc2,
        Cl3 = 'F:',
/* ------------------------------ */

        statement(File, RELSE, Ts, PF,
            [Up3, Low3, X3, Y3, J3, Nc3], ZF),

        PF = [IDF, CELLF, LOCATIONF, SIZEF, cellList(LISTF)|RPF],
        PF2 = [IDF, CELLF, LOCATIONF, SIZEF,
            cellList([cond(Cl3)|LISTF])|RPF],

        Z = [id(File, J1), location(X1, Y1), size(W1, H1)],
        get_line(Z, [ZT, ZF], Connect1),

        P = [id(File, J1), cell(N1), location(X1, Y1), size(W1, H1),
            connect(Connect1), cellList([string(Str1)]), PT2, PF2].
```

図3.2　if then else 文の解析

4．Hichart への変換例

Pascal プログラムから Hichart 記号表現への変換し，図表示するまでを具体例で説明する．

```
program test(output);
label   10;
const   n = 10;
var     i : integer;
        w1, w2 : real;
begin
        read(w1, w2);
        if w1 < w2 then
          writeln('w2 is bigger')
          else
          10: writeln('test');

        for i := 0 to n do
          begin
            w1 := w1 * 2.0;
            w2 := w2 * 2.5
          end;

        repeat
            i := i + 1;
            w1 := w1 * 1.07;
            writeln(w1)
        until  i = n
end.
```

図4.1　Pascal のソースプログラム

（1）はじめに，字句解析によって得られたリストを入力として，構文解析と属性評価を行う．得られる結果の一部を次に示す．

```
 --- grammatical_analyzer_for_pascal
[id(test,0),cell(2),location(50,50 + (50 + 25 + (0 + (50 + 25 + 0
 + (50 + 25) + 0 + 0 + (0 + (50 + 25 + 0 + 1) // 2) + ((50 + 25 +
 0 + 1) // 2 + 50 + 25) + 0 + 0 + 0 + 1) // 2) + 1) // 2),size(10
0,50),connect([connect_code(id(test,0),id(test,1),[[50 + 100,50 +
 (50 + 25 + (0 + (50 + 25 + 0 + (50 + 25) + 0 + 0 + (0 + (50 + 25
 + 0 + 1) // 2) + ((50 + 25 + 0 + 1) // 2 + 50 + 25) + 0 + 0 + 0
+ 1) // 2) + 1) // 2 + 50 // 2],[(50 + 100 + (50 + 100 + 80)) //
2,50 + (50 + 25 + (0 + (50 + 25 + 0 + (50 + 25) + 0 + 0 + (0 + (5
0 + 25 + 0 + 1) // 2) + ((50 + 25 + 0 + 1) // 2 + 50 + 25) + 0 +
0 + 0 + 1) // 2) + 1) // 2 + 50 // 2],[(50 + 100 + (50 + 10· + 80
)) // 2,50 + 50 // 2],[50 + 100 + 80,50 + 50 // 2]]),connect_code
(id(test,0),id(test,1 + 2),[[50 + 100,50 + (50 + 25 + (0 + (50 +
25 + 0 + (50 + 25) + 0 + 0 + (0 + (50 + 25 + 0 + 1) // 2) + ((50
+ 25 + 0 + 1) // 2 + 50 + 25) + 0 + 0 + 0 + 1) // 2) + 1) // 2 +
50 // 2],[(50 + 100 + (50 + 100 + 80)) // 2,50 + (50 + 25 + (0 +
(50 + 25 + 0 + (50 + 25) + 0 + 0 + (0 + (50 + 25 + 0 + 1) // 2) +
 ((50 + 25 + 0 + 1) // 2 + 50 + 25) + 0 + 0 + 0 + 1) // 2) + 1) /
/ 2 + 50 // 2],[(50 + 100 + (50 + 100 + 80)) // 2,50 + 2 * ((50 +
 25 + (0 + (50 + 25 + 0 + (50 + 25) + 0 + 0 + (0 + (50 + 25 + 0 +
 1) // 2) + ((50 + 25 + 0 + 1) // 2 + 50 + 25) + 0 + 0 + 0 + 1) /
/ 2) + 1) // 2) - (50 + 25 + 0 + (50 + 25) + 0 + 0 + (0 + (50 + 2
```

図4.2　解析結果

なお，リストの構造は次の通りである．

[Parent,[Child_1],[Child_2]]

これで，Parent というセルと接続している下位セルが Child_1と Child_2であることが分かる．また次のデータのように

[A,...,connect(B,C,..),...]

connect という述語を含むときは，セルB と C がセルA と線でつながれていることを示す．

図4.2においてセルの座標を求める式，例えば

50+(50+25+(0+(50+25+0)+...

などの生成法を以下に述べる．

- 構文規則を適用した後，変数を用いて属性計算の式を生成する．
- 終端記号，すなわちセルが導出されると，変数に値が単一化され，その値が今度は根の方向に伝搬される．（これに対し，継承属性の場合は単一化された値は根から葉の方向に伝搬される）
- 解析が全て終わると全ての変数に定数が単一化され，定数のみからなる式が生成される．

このようにユニフィケーションを用いて，変数のまま式を構成していき，解析が全て終わったときに定数のみからなる計算式を生成する．そして，式の値が必要になったとき，その式を評価（計算）することができる．これは部分計算と呼ばれ，この機能によりプログラミングが容

易になり，また効率の良い属性評価器が実現できた．

（２）上で得られたリストと接続関係から，各セルの中間表現を生成すると以下のようになる．

```
hichart_code(id(test,0),cell(2),location(50,163),size(100,50),cel
lList([string(['program test'])])) .
hichart_code(id(test,1),cell(15),location(230,50),size(100,50),ce
llList([string([param])])) .
hichart_code(id(test,2),cell(1),location(410,50),size(100,50),cel
lList([string([output])])) .
hichart_code(id(test,3),cell(15),location(230,125),size(100,50),c
ellList([string([label])])) .
hichart_code(id(test,4),cell(1),location(410,125),size(100,50),ce
llList([string(['10'])])) .
hichart_code(id(test,5),cell(15),location(230,200),size(100,50),c
ellList([string([const])])) .
hichart_code(id(test,6),cell(2),location(410,200),size(100,50),ce
llList([string([n])])) .
hichart_code(id(test,7),cell(1),location(590,200),size(100,50),ce
llList([string(['10'])])) .
```

図4.3　セルの中間表現

```
connect_code(id(test,0),id(test,1),[[150,188],[190,188],[190,75],
[230,75]]) .
connect_code(id(test,0),id(test,3),[[150,188],[190,188],[190,150]
,[230,150]]) .
connect_code(id(test,0),id(test,5),[[150,188],[190,188],[190,225]
,[230,225]]) .
connect_code(id(test,0),id(test,8),[[150,188],[190,188],[190,338]
,[230,338]]) .
connect_code(id(test,0),id(test,13),[[150,188],[190,188],[190,451
],[230,451]]) .
connect_code(id(test,1),id(test,2),[[330,75],[370,75],[370,75],[4
10,75]]) .
connect_code(id(test,3),id(test,4),[[330,150],[370,150],[370,150]
,[410,150]]) .
connect_code(id(test,5),id(test,6),[[330,225],[370,225],[370,225]
,[410,225]]) .
connect_code(id(test,6),id(test,7),[[510,225],[550,225],[550,225]
,[590,225]]) .
connect_code(id(test,8),id(test,9),[[330,338],[370,338],[370,300]
,[410,300]]) .
```

図4.4　接続線の中間表現

（３）最後に，得られた中間表現をもとに描画部で Hichart を表示する．

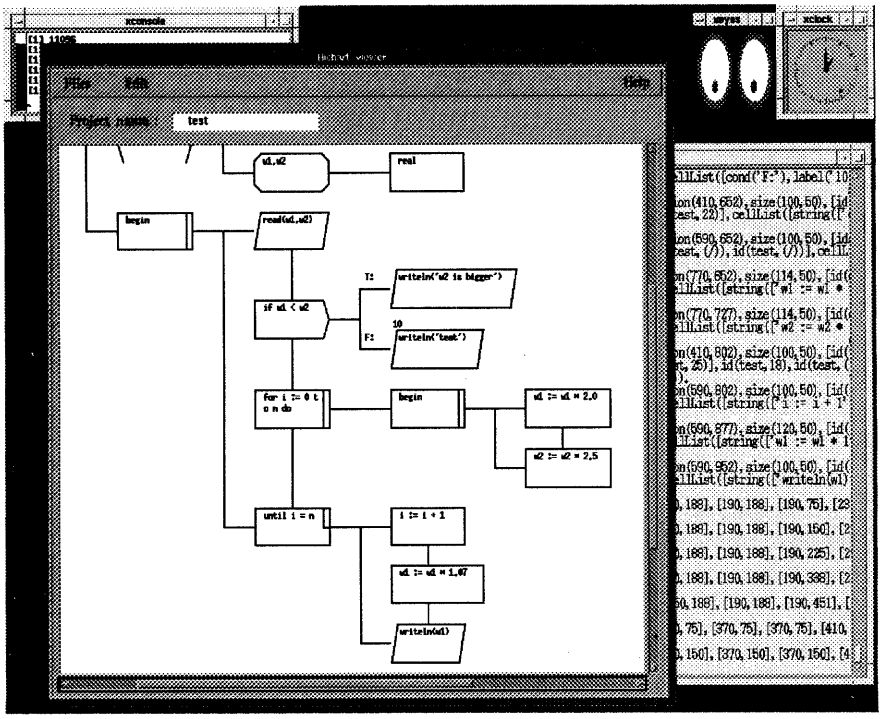


図4.5　Hichart の表示例

5．おわりに

Pascal から Hichart へのトランスレータに関しては，Pascal で記述された Hichart をキャラクタ表示する研究[6]や，C言語で記述された Postscript 言語を利用してグラフィック表示する研究[7]が報告されている．

本研究では，まず ISO 準拠の Pascal から Hichart への変換を属性グラフ文法で定式化し，この文法に基づいてトランスレータを Prolog で実現した．本トランスレータでは，文脈自由文法によるPascal の構文解析と同時に，セルの種類やセル内の文字データなどの属性値を評価している．特に，セルを置く座標については属性値として計算式のみを生成し，必要になったとき値を計算する部分計算の技法を用いて実現した．このように，Prolog は属性文法で記述された処理系を実現するのに適している．

今後の課題として，

- セルの高さが変化しても重ならず，Hichart を見やすく配置するためのセルの美的配置アルゴリズム[例えば[8]]を導入する
- 本論文で定義した属性グラフ文法に基づき視覚的プログラミングツール（構造エディタ）を実現する
- 木構造図用の標準データ交換言語 DXL[9] に対応させる

などが挙げられる．

参考文献

[1] 夜久，他： 階層的流れ図言語 Hichart の情報処理記号，早稲田大学情報科学研究教育センター紀要，Vol.3，pp.92-107（1986）

[2] Yaku,T. et al.:Hichart－A Hierarchical Flowchart Description Language－, Proc. IEEE COMPSAC,Vol.11,pp.157-163(1987)

[3] 西野：属性グラフ文法とその Hichart 型プログラム図式に対するエディタへの応用，コンピュータソフトウェア，Vol.5,pp.81-92（1988）

[4] Nishino,T.:Attribute Graph Grammars with Applications to Hichart Program Chart Editors, Advances in Software Science and Technology,Vol.1,pp.426-433(1989)

[5] ヴィルト(著)，原田(訳)：PASCAL(原著第4版)，培風館(1993)

[6] 郷，他： Hichart プログラム図式の生成手法，情報処理学会論文誌，Vol.31，No.10，pp.1463-1473（1990）

[7] 斎藤，他： PostScript 対応 Hichart 処理系の実現，平成４年度東京電機大学理工学部情報科学科卒業研究概要(1992)

[8] Miyadera,Y. et al.:A Method of Drawing Tree-Structured Program Diagrams on the Euclidian Plane, Proc.IEEE COMPSAC 93, Vol.17, pp.193-201（1993）

[9] 長野，他：木構造図用CASEツール間のデータ交換言語：DXL，情報処理，Vol.35，No.４，pp.341-349（1994）