ソフトウェア工学 105－8
（1995．9．8）

属性グラフ文法に基づいたHichartエディタ

久保 知之[1]　安達 由洋[1]
安斉 公士[2]　土田 賢省[1]　夜久 竹夫[3]

[1] {t-kubo, adachi, kensei}@krc.eng.toyo.ac.jp
[2] anzaik@ibm.net
[3] yaku@loadstar.chs.nihon-u.ac.jp

[1]東洋大学　〒350 埼玉県川越市鯨井2100
[2]関東学園大学　〒373 群馬県太田市藤阿久200
[3]日本大学　〒156 東京都世田谷区桜上水3-25-40

階層型プログラム図式言語 Hichart のためのプログラム図エディタを，Prolog を用いて実現した．このエディタは，属性グラフ文法で定義された Pascal 対応 Hichart の生成規則に基づいて編集コマンドを定義した構文エディタであり，生成や編集過程で構文エラーを引き起こすことはない．そして，レイアウト情報を含む属性をインクリメンタルに評価して Hichart プログラム図を逐次自動描画するという特徴を持ち，未定義の変数などの意味的エラーを検出し表示するなどの編集支援機能も実装している．

この Hichart エディタは，X Window と OFS/Motif を用いた GUI 上に実装されており，統合的ソフトウェア開発支援環境を実現するための重要な視覚的プログラミングツールとなる．

キーワード　Hichart，属性グラフ文法，プログラム図，構文エディタ，インクリメンタル属性評価，視覚的プログラミング環境，

A Syntax-directed Hichart Editor Based on an Attribute Graph Grammar

[1] Tomoyuki KUBO　[1] Yoshihiro ADACHI
[2] Koushi ANZAI　[1] Kensei TSUCHIDA　[3] Takeo YAKU

[1] Faculty of Engineering, Toyo University, 2100, Nakanodai, Kawagoe, Saitama, 350 Japan
[2] Faculty of Economics, Kanto Gakuen University, 200, Fujiagu, Ohta, Gunma, 373 Japan
[3] College of Humanities and Sciences, Nihon University, 3-25-40, Sakurajyousui, Setagaya, Tokyo, 156 Japan

We have developed a syntax-directed editor for producing Hichart program diagrams. Since this editor creates and modifies Hichart diagrams based on an attribute graph grammar of Hichart for Pascal, it never introduces syntax errors in the process of editing. It also has a remarkable feature that, on each operation of editing Hichart diagrams are layouted by evaluating related attributes incrementally. Furthermore, it can find out semantic errors such undeclared-variable and highlight them.

This editor is implemented in Prolog and has easy to handle GUI using X Window and OSF/Motif widgets. It should become an important visual programming tool to realize an integrated software development supporting environment.

key words　Hichart, attribute graph grammar, program diagram, syntax-directed editor, incremental attribute evaluation, visual programming environment

1．はじめに

我々は，Hichart(HIerarchical flowCHART description language)[1,2]のためのプログラム図エディタを，Prologを用いて実現した．このエディタは，属性グラフ文法で定義された ISO準拠のPascalに対応したHichartの生成規則[3]に基づいて編集コマンドを定義した構文エディタ(Syntax-directed Editor)であり，生成や編集過程で構文エラーを起こさない．そして，レイアウト情報を含む属性をインクリメンタルに評価して Hichartプログラム図を逐次自動描画するという特徴を持ち，未定義変数などの意味的エラーを検出し表示する機能も実装している．

このHichartエディタは，X WindowとOFS/Motifを用いたGUI上に実装されており，統合的ソフトウェア開発支援環境を実現するための重要な視覚的プログラミングツールとなる．

2．属性グラフ文法による生成規則の記述

本Hichartエディタの理論的基礎付けに必要な諸概念，文脈自由グラフ文法，属性グラフ文法そしてHichartの生成規則について概説する．

文脈自由グラフ文法[4]　文脈自由グラフ文法は，4つ組$GG=(\Sigma_n, \Sigma_t, P, S)$である．ただし，

(GG1)Σ_nはノードの非終端アルファベットである．

(GG2)Σ_tはノードの終端アルファベットである．

(GG3)Pはプロダクションの集合である．

(GG4)Sは開始グラフである．

なお，プロダクション$p\in P$は以下の条件を満たす4つ組$p=(X, H, I, O)$である．

(P1)$X\in\Sigma_n$

(P2)$H=(V_H, E_H, \phi_H)$は$\Sigma=\Sigma_n\cup\Sigma_t$上のグラフ

(P3)$I\in V_H$を入力ノードという．

(P4)$O\in V_H$を出力ノードという．

ここで，V_Hはノードの有限集合，E_Hは辺の有限集合，$\phi_H: V_H\rightarrow\Sigma$はnode labeling functionである．ノードxが$\phi_H(x)=X$であるとき，単にノードXとも呼ぶ．また$Lab(H)=\{X \mid x\in V_H, X=\phi_H(x)\}$とする．

プロダクション$p=(X, H, I, O)$を用いたグラフGからG'への書き換え(導出ともいう)は，以下の一連のステップを通じて行われる．

(step1)Xとラベル付けされたノードxをグラフGから除去する．

(step2)xのあった位置にグラフHと同形なグラフH'を置く．このとき，HからH'への同形写像をfとする．(新しいノード名への付け替えのため)

(step3)もしxの入力辺が存在するならば，それは$I'=f(I)\in V_{H'}$に接続される．

(step4)もしxの出力辺が存在するならば，それは$O'=f(O)\in V_{H'}$に接続される．

グラフGとプロダクションpによって得られたグラフG'が存在するとき，GからG'が直接導出されたといい，$G\underset{p}{\Rightarrow}G'$とかく．

$G_0\underset{p_1}{\Rightarrow}\cdots\underset{p_n}{\Rightarrow}G_n$であるようなプロダクション列$\mathbf{p}=(p_1, p_2, \cdots, p_n)$が存在するとき$G_0$から$G_n$が導出されたといい，$G_0\underset{\mathbf{p}}{\Rightarrow}G_n$とかく．

属性グラフ文法[5]　属性グラフ文法は3つ組$AGG=(GG_u, A, F)$である．

(AGG1)$GG_u=(\Sigma_n, \Sigma_t, P, S)$は，AGGの基底文脈自由グラフ文法である．

(AGG2)GG_uの各ノードアルファベット$X\in\Sigma$に対し，互いに素な2つの有限集合，継承属性の集合I(X)と合成属性の集合S(X)が付随している．Xの属性全体の集合を$A(X)=I(X)\cup S(X)$で表す．$A=\cup_{x\in\Sigma}A(x)$をAGGの属性集合という．

(AGG3)Pの各プロダクション$p=(X, H, I, O)$に対し，$S(X)\cup(\cup_{X_r\in Lab(V_H)}I(X_r))$の属性のみをすべて定義する意味規則の集合$F_p$が付随している．集合$F=\cup_{p\in P}F_p$をAGGの意味規則集合という．

Hichartの生成規則[3]

属性グラフ文法により，Hichartの生成規則を定義する．Hichartでの，終端アルファベットおよび非終端アルファベットの一部を表1に，継承属性および合成属性の一部を表2に示す．図1は属性グラフ文法で定義されたPascal対応Hichartの生成規則と意味規則の抜粋を示したものである．

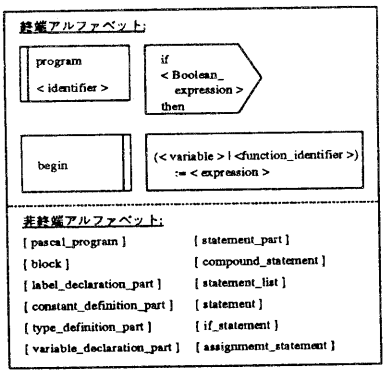

表1　Hichart生成規則でのアルファベット

継承属性:	
top	：部分木内で最も上に配置されるセルのY座標
x	：配置されるセルのX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	：セルの最小の幅、最小の高さ
GapX, GapY	：垂直方向と水平方向における隣接セルの間隔
id	：セルの識別番号
合成属性:	
bottom	：部分木内の最下辺のY座標
y	：配置されるセルのY座標
w, h	：セルの幅、高さ
cell	：セルの種類(番号)
string	：セルの内部の文字列
lines	：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

表2　Hichart生成規則での属性

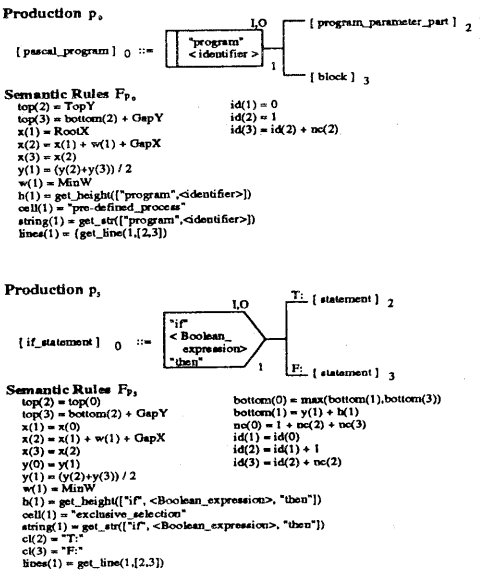

図1　生成規則と属性評価式(意味規則)

3．コマンドのモデル化

エディタコマンドをHichartの生成規則を用いて形式的にモデル化する．生成，挿入および削除のコマンドが，生成規則に基づいて定義されているので，このエディタは構文的に正しいHichartしか生成しない．

3.1　プロダクション列の操作

導出$G \underset{p}{\Rightarrow} G'$において，プロダクション・インスタンス(以後，単に"インスタンス"と書く)とは組$i=(w, p)$のことである．ただし，pはプロダクションであり，$w \in V_G$は導出過程で除去されたノードである．インスタンス$i=(w, p)$によって，グラフGがグラフG'を直接導出することを$G \overset{w}{\underset{p}{\Rightarrow}} G'$と書く．

$\mathbf{p}=(p_1, p_2, \cdots, p_n)$をプロダクション列，$(w_i, p_i)$を各プロダクション$p_i$に対するインスタンスとする．このとき，列$((w_1, p_1), (w_2, p_2), \cdots, (w_n, p_n))$をインスタンス列という．

Sを開始グラフ，G_nをSから導出されるグラフとし，その導出列を $S \overset{w1}{\underset{p1}{\Rightarrow}} G_1 \overset{w2}{\underset{p2}{\Rightarrow}} \cdots \overset{wi-1}{\underset{pi-1}{\Rightarrow}} G_{i-1} \overset{wi}{\underset{pi}{\Rightarrow}} G_i \overset{wi+1}{\underset{pi+1}{\Rightarrow}} \cdots \overset{wn}{\underset{pn}{\Rightarrow}} G_n$とする．

このとき，もし$G_{i-1} \overset{w}{\underset{q}{\Rightarrow}} Q$であるようなプロダクション$q=(X_q, H_q, I_q, O_q) \in P$が存在し，かつ，$G_{i-1} \overset{w}{\underset{q}{\Rightarrow}} Q \overset{w'}{\underset{pi}{\Rightarrow}} G'_i \overset{wi+1}{\underset{pi+1}{\Rightarrow}} \cdots \overset{wn}{\underset{pn}{\Rightarrow}} G'_n$であるような導出列が存在するならば，qは$p_i$で挿入可能であるという．

さらに，プロダクション$q=(X_q, H_q, I_q, O_q)$がp_iで挿入可能であり，かつ，G_iに適応できる任意のインスタンス列がG'_iにも適応できるときqはp_iに真に挿入可能であるという．

プロダクション$q=(X_q, H_q, I_q, O_q)$で$X_q = \phi_{Hq}(w')$であるようなノード$w' \in V_{Hq}$が存在すれば，qはp_iで真に挿入可能となる．

3.2　挿入コマンド

G_nを現在のグラフ，ノード$x \in V_{Gn}$を指定されたノードとする．また，導出列$S \Rightarrow \cdots \Rightarrow G_{i-1} \Rightarrow G_i \Rightarrow \cdots \Rightarrow G_n$において，$G_i$をノードxがはじめて出現したグラフとする．

プロダクション$q=(X_{pi}, H_q, I_q, O_q) \in P$が$p_i$で真に

挿入可能であるとき，インスタンス(w_i, q)のインスタンス列$((w_1, p_1), \cdots, (w_n, p_n))$への挿入とは，インスタンス列$((w_1, p_1), \cdots, (w_{i-1}, p_{i-1}), (w_i, q), (w', p_i), \cdots, (w_n, p_n))$を作ることである．ただし，インスタンス$(w_i, p)$の適用によりノード$w_i$を置き換えたグラフを$H'_q$としたとき$w' \in V_{H'_q}$, $\phi_{H'_q}(w') = X_{p_i}$とする．

これに対応するグラフの導出過程は以下のようになる．

(1)導出列をG_nからG_{i-1}まで戻る．

(2)G_{i-1}に対してインスタンス(w_i, q)を適用し，グラフQを得る．

(3)Qに対して$((w', p_i), (w_{i+1}, p_{i+1}), \cdots, (w_n, q_n))$を適用し，$G'_n$を得る．

この導出で得られたG'_nはG_nのノードxが出現する直前にグラフH'_qが挿入されたグラフとなっている．すなわち，このインスタンス列へのインスタンスの挿入は，Hichart図へのセルの挿入に対応している．そして，真に挿入可能であるという条件は，Hichartエディタでセルが挿入できるための条件に対応している．

セルあるいは部分プログラム図の削除についても，挿入と同様にプロダクションインスタンス列の操作により形式的に定義される．

4．Hichartエディタでの編集

ここでは，本Hichartエディタを用いたHichartプログラム図の作成過程を説明する．

4.1 GUI上のプログラム図表示とその操作

Hichartのプログラム図は，セルと呼ばれるプログラムの制御と処理を表現する記号，セルを結ぶ線，そしてセルの内側と外側に記述される文字列であるフレーズによって構成される．

本Hichartエディタでは，このセル，フレーズ，および生成規則の右辺に対応したHichartの部分図であるテンプレートをユーザが編集できる対象の基本要素とする．プログラム図の生成および編集操作を対象とするエディタの場合，終端ノード（終端セルと呼ぶ）の記号だけではなく，非終端ノード（非終端セルと呼ぶ）の記号も図形データとして表示しなければならない．ユーザは非終端セルをテンプレートに置き換える操作，すなわち生成規則の適用を繰り返すことによりHichartプログラム図を生成していく．

図2はHichartエディタの初期画面であり，'program'および'begin'という文字列が記入されているセルは終端セルである．図中において色が濃く，[program parameter part]，[label declaration part]などの文字列が記入されているセルは非終端セルである．

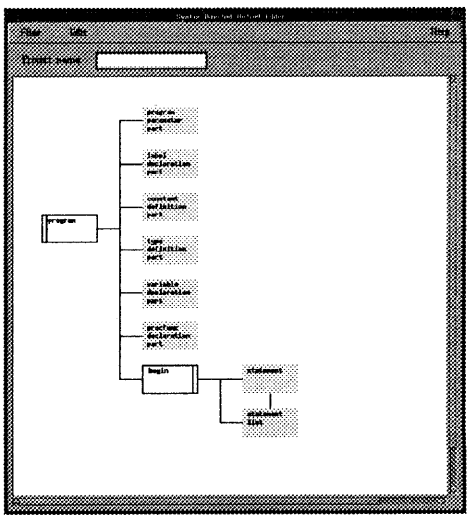

図2　Hichartエディタの初期画面

GUI上での非終端セルを終端セルに置き換える操作について説明する．Hichartプログラム図は非終端セルをテンプレートに繰り返し置き換えることにより，トップダウンに生成される．もし[program parameter part]とラベル付けされた非終端セル上にマウスを動かしクリックすると，非終端セルが生成規則に基づき置き換えられ，結果として図3のようなHichartプログラム図が表示される．

[statement]とラベル付けされた非終端記号の上でマウスをクリックしたとき，図4に示す非終端セルを置き換えることのできる候補を表示したポ

ップアップウインドウが即座に現れる.

プッシュボタン'if'をクリックすると，すぐにポップアップウインドウは消え，非終端セルが図5に示すif文に対応する生成規則の右辺に書き換えられる.

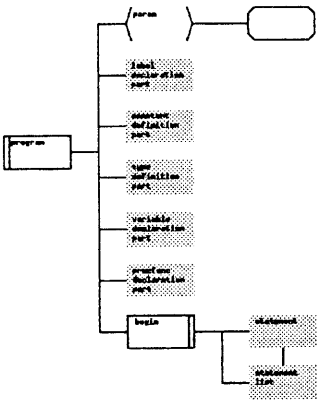

図3　生成規則適用後のHichartプログラム図

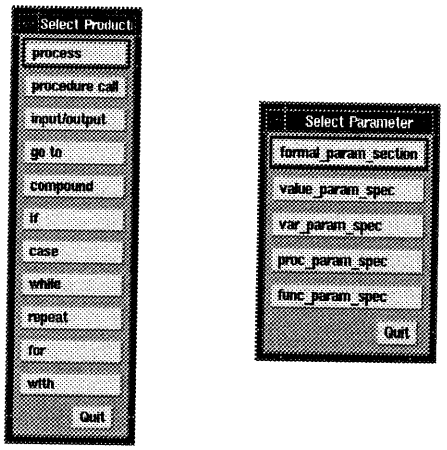

図4　選択用ポップアップウインドウ

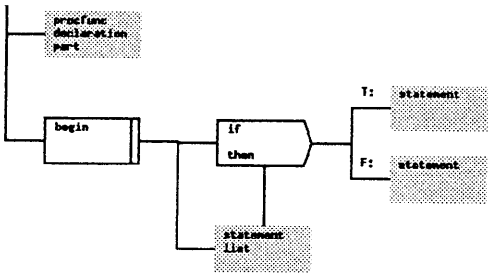

図5　[statement]のif文への置き換え

上記のように本Hichartエディタでは，各生成規則を逐一適用してはいない．それは生成規則を1つ1つ適用すればユーザに非常に手間どらせる結果となるからである．このため，本Hichartエディタでは効率のよい操作のために一連の生成規則を作業の単位として扱う.

多くのプロダクションを実行した後，図2に示される初期Hichartプログラム図が生成される.
同様に置き換えのためのテンプレートも，いくつかの一連のプロダクションを適用した後に生成される部分プログラム図を用意している.

次にフレーズの操作に関しては，文字列が記入できる終端セルの内側をマウスでクリックするとセル上にテキストウインドウが生成され（テキストウインドウが存在するかしないかはテキストカーソルが存在するかしないかで確認することができる），ユーザはテキストウィジェットを通して任意の文字列を入力することができる.

4.2　挿入

ここではHichartエディタ上での挿入の例として，図6に示されるHichart部分プログラム図の文字列'x:=1'と'z:=3'を持つセルの間に[statement]とラベル付けされた非終端セルの挿入について説明する.

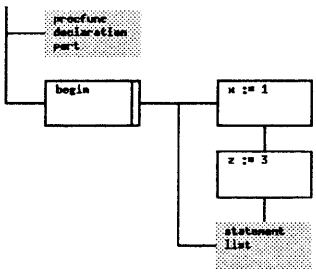

図6　Hichart部分プログラム図の例

プルダウンメニューのボタンをクリックすることで挿入コマンドを呼び出した後，文字列'z:=3'を持つ終端セルの内側をマウスでクリックする．その後，[statement]とラベル付けされた非終端セルが図7のように挿入される.

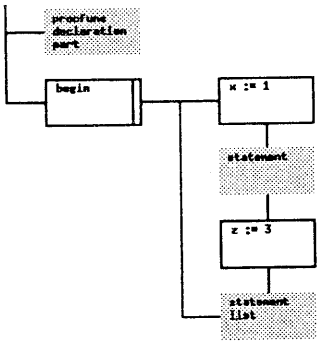

図7　挿入後のHichart部分プログラム

4.3　削除

削除コマンドも挿入の場合と同様に，プルダウンメニューのボタンをクリックすることにより呼び出される．図6に示されるHichart部分プログラム図において削除コマンドを呼び出し，文字列'x:=1'を持つ終端セルの内側をマウスでクリックした後，結果として図8のようなHichart部分プログラム図になる．

編集を行う間，本エディタは構文エラーを引き起こすことはない．また，未宣言変数のような意味的エラーは発見され，それを含むセルはハイライトされる．

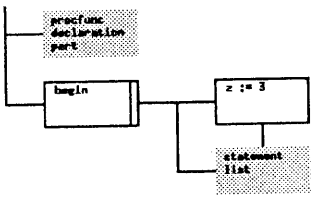

図8　削除後のHichart部分プログラム図

5．実装

本Hichartエディタには，ISO標準Pascalに対応したHichartプログラム図を生成するインクリメンタルな属性評価の機構を実装した．それは図9のような導出木を用いてインクリメンタルに属性を評価し，未宣言変数のような意味的なエラーを見つけるだけではなく，一定のレイアウト条件に基づくHichartプログラム図のレイアウトをインクリメンタルに行う．

ここではHichartプログラム図の内部データ表現とエディタコマンドを実装する方法を記述する．

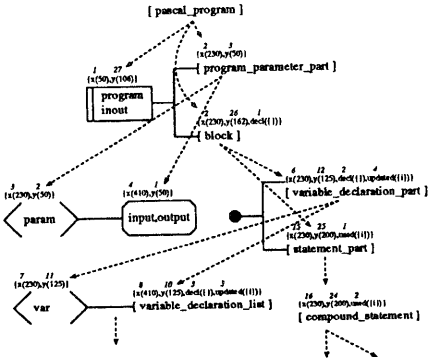

図9　Hichartの導出木をその属性の一部

5.1　Hichartプログラム図の記号表現

Hichartエディタでは，終端セルだけでなく，非終端セルのデータも読込み，保存および表示ができなければならない．そのため本研究では，[3]で報告されているHichartプログラム図の記号表現を拡張し，各々のセルをPrologの項として以下のように表現した．

```
extended_hichart_code(ID, Cell, location(X, Y),
    size(W, H), link(Parent, Children, Previous, Next),
      cellPhrase(Phrase), Attribute).
```

各々の引数の意味は以下の通りである．

ID：セルの識別番号
Cell：セルのタイプ
location：セルのxとy座標
size：セルの幅と高さ
link：隣接セルへのリンクポインタ
cellPhrase：セルの内側または外側に書かれるフレーズ
Attribute：プログラム図のデータを除いた属性で例えば変数の宣言など

セルの識別番号は，Hichartプログラム図の記号表現においてユニークであり，リンクポインタは，Hichartプログラム図の生成，編集において不可欠である．

テンプレートが例えば[label declaration part]またはif文のelse部などの省略できる非終端セルを含むとき，エディタは省略しないで全てのテン

プレートを表示する．そして内部記号表現から対応するPascalプログラムに変換されるとき，省略できる非終端セルが省略される．

セルに接続する各線は以下のように表現される．

extended_connect_code(ID1, ID2, Linedata).

ここで，ID1とID2は接続するセルの識別番号で，Linedataは[[x1, y1], [x2, y2], ・・・]のようなリストである．また，各[xi, yi]は線が通過する各点のXおよびY座標である．

5.2 各コマンドのアルゴリズム

Hichartエディタの各コマンドは，3節で示したように生成規則に基づいて形式的に定義されている．一方，コマンドを実行する過程は，コーネルプログラムシンセサイザー[6, 7, 8]において示されたように導出木における部分木の置き換えとしてみることもできる．

そこで，エディタコマンドの実現は，導出の履歴を用いないで部分プログラム図のパターンマッチ，セルの削除，セルの追加，リンクの付け替えおよび属性評価などの基本手続きを用いて行った．ただし，挿入および削除のための条件チェックはコマンドの形式的定義に基づいて厳密に実行される．

各コマンドは，ユニフィケーションと呼ばれるパターンマッチング能力と再帰的プログラミングなどPrologの持つ特徴を利用してプログラミングされている．ここではPrologによる代表的なコマンドの実装を示す．

生成規則（言い換えれば，非終端セルの対応するテンプレートへの置き換え）を適用するためのプログラムの概略は以下のようである．

```
procedure REPLACEMENT(Cell);
  {Attr:Attribute, Temp:Template}
procedure delete_routine(Temp);
  begin
    Root = get_rootCell(Temp);
    {get lines connected with the cell}
    get_lines(Root, Lines);
    delete_lines(Lines);
    delete_template(Temp)
  end;
procedure insert_routine(Temp);
begin
  insert_template(Temp);
  Root = get_rootCell(Temp);
  connect_line(Root)
end;
procedure replace_routine(O_temp, N_temp);
begin
  delete_routine(O_Temp);
  Size = get_size(N_Temp);
  {move the affected cell and lines}
  LAYOUT(N_Temp, Size);
  insert_routine(N_Temp)
end;
begin
  Attr = get_attribute(Cell);
  select_procedure_routine(Attr, Temp);
  replace_routine(Cell, Temp)
end.
```

図10 アルゴリズム REPLACEMENT

挿入と削除についても上記と同様の方法で実装した．

本エディタでは，Hichartを分かり易く表示するために以下のレイアウト条件を考慮している．

条件1 プログラムの階層構造により，同じ階層レベルにあるセルは水平方向で同じ座標に配置される．

条件2 垂直方向において，隣接した部分木間のスペースの幅は一定である．

条件3 親セルはその子セルの真ん中に配置される．この条件は，[statement_list]とラベル付けされた非終端記号を持つプロダクションの場合は例外である．この場合，[statement list]に対応する部分木のルートセルと親セルは垂直方向に同じ座標に配置される．

このレイアウトを実行するプログラムの概略は次のようである．

```
procedure LAYOUT(Template, Size);
  procedure layout_routine(i);
  begin
    {For Ci∈Path and Ci has the level i,
     let αi=y'(Ci)-y(Ci), where y'(Ci) is newly
     placed by the procedure LAYOUT.
     Compute αi-1=y'(Ci-1)-y(Ci-1) as follows. }
    if the cell Ci-1 has only the child Ci then
      αi-1=αi
```

```
    else {The cell C_{i-1} has more than or equal to
          2 children.}
      Let N_i=#{P|P is the brother of C_i,
      y(P)<y(C_i)},
      α_{i-1}=(α_i+β)/2^{N_i+1}
    end;

begin
  {Compute a new y-coordinate y'(P) of each cell P
   when the cell C_L with level L is replaced by
   Template.}
Path := C_L∪{a set of ancestor of C_L};
Above := {Q|C∈Path,[x(C),x(C)+width(C)]∩
  [x(Q),x(Q)+width(Q)]≠φ,y(C)>y(Q)};
Below := {R|C∈Path,[x(C),x(C)+width(C)]∩
  [x(R),x(R)+width(R)]≠φ,y(C)<y(R)};
ABOVE := Above∪{S|S has an ancestor in Above};
BELOW := Below∪{T|T has an ancestor in Below};
if X∈ABOVE then
  y'(X) := y(X);
if X∈BELOW then
  y'(X) := y(X)+β;
LABEL := get_RootLabel(Template);
if LABEL == 'begin' then
  begin_layout_routine(Template,Size);
else if LABEL == 'repeat' then
  repeat_layout_routine(Template,Size);
else
  for i := L downto 1 do
    layout_routine(i)
end.
```

図１１　アルゴリズム LAYOUT

手続きLAYOUTは属性グラフ文法の意味規則に埋め込まれ，描画は$O(n)$時間で行われる．ここでnはノード（セル）の数である．

６．おわりに

属性グラフ文法で定義されたISO標準Pascalに対応したHichartの生成規則に基づいて，Hichart図を生成および編集するコマンドを形式的に定義し，これをもとにHichartプログラム図のための構文エディタをPrologで実現した．

このエディタは，入力の順序などのある程度の柔軟性を持たせた編集機能を持ちながら，文法エラーを発生させない構文指向を実現したエディタとなっている．

エディタコマンドは，Prologの持つ特徴を利用して部分プログラム図のパターンマッチ，セルの削除，セルの追加およびリンクの付け替えなどの基本手続きを用いて実装した．ただし，挿入および削除のための条件チェックはコマンドの形式的定義に基づいて厳密に実行される．

また，本エディタにはレイアウトのアルゴリズムも組み込まれている．このアルゴリズムは属性文法によって表現されるレイアウト条件に基づいているため直感的でわかりやすい形で実装され，編集操作に伴って自動的にかつインクリメンタルに実行されていく．このレイアウトルーチンは他のレイアウト条件に基づくルーチンによって容易に置き換えることが可能である．

なお，本システムはIF/PrologとOFS/Motif1.2で記述され，SunOS4.1.3およびSolaris2.4上で稼働している．

参考文献

[1]夜久，他：階層型流れ図言語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,(1986),92-107

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

[3]大井，他：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，電子情報通信学会技術研究報告，COMP94-100,(1995)90-96

[4]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,(1988),81-92

[5]Vigna,P.D. et al.: Context-Free Graph Grammars, Inf. Control, Vol.37,(1978),207-233

[6]Teitelbaum,T.et al.:The Cornell Program Synthesizer:a syntax-directed programming environment. Communications of the ACM 24,(1981),563-573

[7]Demers,A. et al.:Incremental Evaluation for Attribute Grammars with Application to Syntax-directed Editors. Conference Record of the Eighth ACM Symposium on Principles of Programming Languages,(1981),105-116

[8]Reps,T.:Optimal-time Incremental Semantic Analysis for Syntax-derected Editors. Tech. Report No.81-453, Dept. of Comptr. Sci., Cornell University, Ithaca, NY,(1982),169-176