卒業論文

Hichartエディタの Javaによる実現
Implementation of Hichart Program Diagram Editor by Java

永田啓介
Keisuke Nagata

あらまし 本論文では, C 言語のソースプログラムを Hichart に変換させるプログラムを Java 言語で実装する事を目的とする. これにより特定のプラットホームに依存することなく動作する.

はじめにエディタコマンドに対応する属性グラフ文法について解説する. 次に Hichart の解説, そして Hichart を表現するための汎用データファイルである H-code2 を解説し, 最後にに C-Hcode2 トランスレータについて述べる.

キーワード Hichart, 属性グラフ文法, Java, H-code2, C-Hcode2 トランスレータ

1 はじめに

本論文では, Hichart エディタの定義である属性グラフ文法と属性文法を定義し, これに基づき, 最後に C のソースプログラムを Hichart で図表示するための Hichart エディタを Java 言語で実装することにより, Hichart エディタを幅広く, そして簡単にユーザーに取り扱えるようにする.

プログラムの図表示はプログラム開発の際, 全体構造が即座に把握でき, 部分的な修正, 変更, トレースも容易に可能となる. そこで近年, ノイマン型フローチャート, NS チャートを筆頭に, それぞれの特徴を持つ多様のプログラム図式の記述言語が報告されている. その１つが Hichart フローチャートである. [1][3]

１９９５年安達他により属性グラフ文法に基づく構文指向 Hichart エディタが開発された. [3]

Hichart エディタは ISO 標準 Pascal 言語に対応した Hichart 図を作るもので, 文脈自由言語の記述が容易であるなどの理由のため, IF/Prolog 言語で実現されているが, Prolog 処理系は環境が整ってないと使用できないなどと動作環境が限定されて, 一般的であるとは言い難い. 一方 Java で書かれたプログラムは特定のプラットホームに依存することなく実行できるという利点を持つ. これにより幅広く, 簡単に取り扱えるようにする. [5][6][7][8]

2 諸定義 [3][4][5][6][7][8]

Hichart は属性グラフ文法により定式化される. 属性グラフ文法は文脈自由グラフを拡張して作られた拡張文法である. すなわち, 文脈自由グラフ文法の各節記号に属性を付加し, 属性の値を計算するための意味規則を各プロダクションに付加することで作られる.

2.1 グラフ

Σ (ノードアルファベット (node alphabet)) 上のグラフ⇔３項組 H=(V,E,φ) 但し,

- V はノード (node) の空でない集合

- E $\subseteq$ V $\times$ V はエッヂ (edge) の集合
- φ:V $\rightarrow \Sigma$ はノードラベリング関数 (node labeling function) という.
  $\exists$v $\in$ V, $\exists$x $\in \Sigma$ で x=φ(v) となっているとき, v は x とラベル付けされたといい, x を v のラベルという.

2.2 文脈自由グラフ文法

文脈自由グラフ文法はグラフ導出に関するものを形式的に記述したものである. Hichart では, 一つ一つのセルを形式的に生成していく方法に文脈自由グラフ文法が応用されている.

文脈自由グラフ文法 (context-free graph grammer)(CFGG) とは, 5 項組 GG=(Σ_n, Σ_t, S, D_0, R) である. 但し,

- Σ_n は非終端ノードアルファベット (non terminal node alphabet)
- Σ_t は終端ノードアルファベット (terminal node alphabet)
- S $\in$ Σ_n は開始ラベル
- D_0=(V_0, E_0, φ_0) は開始グラフ (start graph). 但し, V_0=v_0, E_0=ϕ , φ_0(v_0)=S
- R はプロダクション (production rule) の集合, R の要素は 4 項組 r=(A, D, I, O) で表される. 但し,
  - A $\in$ Σ_n
  - D=(V, E, φ) はΣ=Σ_n $\cup$ Σ_t 上の連結グラフ
  - I $\in$ V は入力ノード (input node)
  - O $\in$ V は出力ノード (output node)

プロダクション r=(A, D, I, O) は A $\rightarrow D^{I,O}$ と書くことができる. また, 入力ノードと出力ノードが明らかな時は, A $\rightarrow$ D と書く. A をプロダクションの左側, D を右側という. いくつかのプロダクション r によって,S とラベルされたひとつのノードのみからなるグラフ D_0(開始グラフ) から始めて, いくつかのプロダクションを適用して終端ノードアルファベットによってラベルがされたグラフが導出される.

導出は次のようにおこなわれる. 書き換えられる前のグラフを D_1, 適用するプロダクション r を A $\rightarrow D^{I,O}$ とすると, D_1 の非終端ノードアルファベットでラベルされた (例えば A でラベルされた) ノードが, D と同型なグラフで置き換えられ, グラフ D_2 となる. もともと v に入っていた辺は I に入る辺となり, v から出て行く辺は 0 から出て行く辺となる D_1 と D_2 の関係を $D_1 \Rightarrow D_2$ と書き, D_1 から D_2 が直接導出されるという. また, r をノード v に適用したプロダクションという.

$\Rightarrow$の推移閉包を $\overset{*}{\Rightarrow}$ とする. G$\overset{*}{\Rightarrow}$H のとき, G から H が導出されるという.

2.3 属性グラフ文法

属性グラフ文法は文脈自由グラフを拡張し, 一つ一つのノードに二つの属性を付随させたものである. Hichart では, 先に定義したセルの生成に関する文脈自由グラフ文法をふまえ, 各々のセルの形や配置の仕方を一般化したものが属性グラフ文法となる.

属性グラフ文法 (Attribute Graph Grmmar) は次の条件を満たす 3 項組 G=< GG, A, F >である.

- GG=(Σ_n, Σ_t, S, D_0, R) は AGG の基底文脈自由グラフ文法と呼ばれる. また, Σ=Σ_n $\cup$ Σ_t 上のグラフ D=(V, E, φ) について, Lab(D)={ φ(n)| n $\in$ V} とする.
- GG の各元 X $\in \Sigma$に対して, 互いに素な 2 つの有限集合である, 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.

$$A=\bigcup_{X \in \Sigma} A(X)$$

を G の属性集合という. 但し, I(S)=ϕ , X $\in$ Σ_t に対しては S(X)=ϕ とする. また, X の属性 a を a(X) で表し, a がとりうる値全体の集合を V(a) で表す.

- R の各プロダクション r=$X_0 \rightarrow$ D に対し,

$$S(X_0)=\bigcup_{X \in Lab(D)} I(X)$$

の属性のみを全て定義する意味規則の集合 F_r が付随している. 属性 $a_0(X_{i_0})$ を定義する意味規則は, $a_0(X_{i_0})$:=f($a_1(X_{i_1})$, $\cdots$, $a_m(X_{i_m})$), $0 \leqq i_j \leqq$ |Lab(D)|, $X_{i_j} \in$ Lab(D),0 $\leqq$ j $\leqq$ m という形である. f は V($a_1(X_{i_1})$) $\times \cdots \times$ V($a_m(X_{i_m})$) から V($a_0(X_{i_0})$) の中への写像である. この時, $a_0(X_{i_0})$ は r において $a_j(X_{i_j})$(1 $\leqq$ j $\leqq$ m) に依存するという. 集合

$$F=\bigcup_{r \in R} F_r$$

を AGG の意味規則集合という.

3 Hichart[2][3]

3.1 Hichart

木構造型プログラム図言語 (木フローチャートを記述するための言語). Hichart フローチャートは, 従来の Goldstein-Neumann 型のフローチャートにおける基本記号と制御線をそのまま残して, 制御記号 (反復や分岐など) を変えたり, 追加することによりプログラムの使用を形作る 4 つの基本要素,

- アルゴリズムの流れ
- データの流れ
- データ構造
- プログラム構造

を全て一様に表示することが可能になった. そのため視認性と記述性が飛躍的に向上し, 全体が即座に把握できるようになっている. Hichart 図は

- 制御や処理を表す記号 (セル)
- セルとセルを接続する線 (接続線)
- セルの内側, あるいは外側にある文字列

の 3 つの要素からなっている.

Hichart の構文は属性グラフ文法により定式化されている. その生成規則は 67 個である. また, Hichart 図の描画に関しては, 美的描画の理論に基づき属性によって計算することが出来る.

3.2 セルについて

- 汎用処理記号
  あらゆる種類の処理, 型, 機能, 装置を表す.

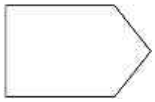
- 定義済み処理セル
  処理, 型, 機能, 装置などの定義を表す. また別の場所で定義されているそれらの参照を表す.

- 見出しセル
  セルの内側の部分図の種類を表す. すなわち, プログラム流れ図, データ構造図などの別を明記したいときに用いる.

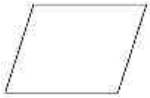
- 名前セル
  データ, 手続き, システム等の名前を表し, それらを強調したい場合に用いられる. また変数の型を宣言するときに必ず用いる.

- 連続逐次反復セル
  無限回反復, 回数があらかじめ指定された順次反復を表す.

- 択一選択分岐セル
  セルの右側の部分図が条件に従い, 高々一つだけ選択されることを示す.



- 汎用データセル
  媒体を指定しないデータとその入出力処理, 入出力機能を表す.



4 H-code2[2][9]

H-code2 とは, Hichart 処理系における Hichart フローチャートの内部表現のための言語 (中間言語).

4.1 仕様

- １つのファイルで構成されている.
- テキストファイルである.
- 拡張 BNF を用いてプログラミング言語風に定義されている.
- Hichart の階層構造に対応した階層構造になっている.
- 上記により人間が容易に理解でき, 編集可能である.
- セルの種類を表す数値やテキストはセルの種類ごとに分類し, 定義されている.
- セルに最低限指定しなければならない情報がセルの種類とセル内部に表示する文字列だけなので,H-code と比較してファイルサイズの縮小が望める.
- １つのファイル中に複数の Hichart が収納可能であり, さらにそれぞれの属性, 関係が表現可能である.
- さまざまなデータスコープで H-code2 ファイルで用いる数値の倍率, 単位など (拡張データ) が指定可能である.

4.2 文法

(1) ヘッダ
H-code2 ファイル全体に関する情報が書き込まれていて, 図 1 に示すような構造である.

- H-code2 ファイルが作成された日付, 時間を記述する.
- H-code2 ファイルを作成または更新したアプリケーションの名前とバージョンを記述する. これらのデータを参照することで Hichart 処理系が容易に H-code2 ファイルの処理が容易になる可能性が高くなる.
- オプションとして拡張データを記述する. ここに記述された拡張データは H-code2 ファイルに含まれる全ての Hichart に影響を与える. そのため, 全ての Hichart に共通する指定はここに記述することにより, 変更などが容易になる.

```
"header{"
     year '/' month '/' day
     hour ':' minute ':' second
     application_name
     version
     [extended_data]
'}'
```

図 1 : ヘッダの構造

(2) 木ヘッダ
ひとつの Hichart に固有の情報が書き込まれて, 図 2 に示すような構造である.

- 木の名前を記述する. これは他の木またはセルから参照される場合に用いられる.
- 木の作成または更新された日付, 時間, アプリケーションの名前とバージョンを記述する. これらの使用法はヘッダのものと同じ.
- 木の種類を記述する. 例えば木が Pascal のプログラムを表すならば, Pascal と記述する. また, 木がプログラム流れ図ならば, program flow と記述する. これらの記述を組み合わせて Pascal program flow のようにも記述可能である. このデータにより Hichart 処理系は, 木が自分が処理できる木かどうかが容易に確認できる. 次の木の接続情報は, 複数の木が関係を持っている場合に用いられるものである. 意味は処理系に依存している.
- オプション拡張データが記述できる. ここに記述された拡張データは 1 つの木のみに影響を与える.

```
"tree{"
     "tree_header{"
           [tree_name]
           year '/' month '/' day
           hour ':' minute ':' second
           application_name
           version
           tree_type
           parent_tree_connection
           older_tree_connection
           younger_tree_ connection
           child_ tree_ connection
           [extend_data]
     '}'
     [tree_data]
'}'
```

図 2 : 木ヘッダの構造

(3) セルデータ

セルを表すデータ, 図 3 に示すような構造である.

- セルの名前を記述できるが, これも木の名前同様に他のセルからの参照に用いられる.
- セルの種類を表す数値, セル内部に描かれる文字列を指定する. その他の指定がしたい場合は次の拡張データで指定する. ここに記述される拡張データは 1 つのセルのみ, またはこのセルをルートセルとする部分木全体に影響を与える. この両者は拡張データを表す文字列の先頭で区別される. 拡張データが"ALL"で始まっていれば, 部分木全体に影響を与える拡張データである. もしセルが子供を持っているならば拡張データの次に入れ子にして記述する.
- セルと次の弟セルとの間に制御の移動がなければ, ',' を記述する.

```
'{'
    [cell_name]
    class
    cell_string
    [extended_data]
    [tree_data]
'}'
[',']
```

図 3 : セルデータの構造

5 C-Hcode2 トランスレータ

C-Hcode2 トランスレータとは, C 言語のソースプログラム (ポインタや構造体などの複雑なものは除く) を H-code2 に変換するプログラムである. この C-Hcode2 トランスレータは Java 言語で実装されたことにより, プラットホームに依存せず実行できる. また, オブジェクトとして扱うことが出来る.

この C-Hcode2 トランスレータは複数のファイルで成り立っている. その個々のファイルの動作を以下で解説する. (例 : 図 5-1, 図 5-2, 末尾の付録参照)

- compiler94.java
  H-code2 のヘッダ部分を作り, "header" ファイルとして出力する.
- compilerHT41X.java
  C 言語のソースプログラムを読み込みある程度 H-

code2 のかたちを作る. Main 関数の部分を”main”ファイル, ほかの部分を”etc”ファイルとして別々に出力する.

- compiler52X.java
  ”main”ファイルと, ”etc”ファイルのなかみを結合する. このとき Main 関数が最後になるようになっている. この結果を”body”ファイルとして出力する.

- compiler11X.java
  ”body”ファイルを読み込み入れ子の深さに対応させて字下げする. また, 並列処理のところには”,”をつける. この結果を”fullbody”ファイルとして出力する.

- compiler51X.java
  ”header”ファイルと”fullbody”ファイルを読み込み, 結合し, 少し手を加え”Hcode2”ファイルとして出力する.

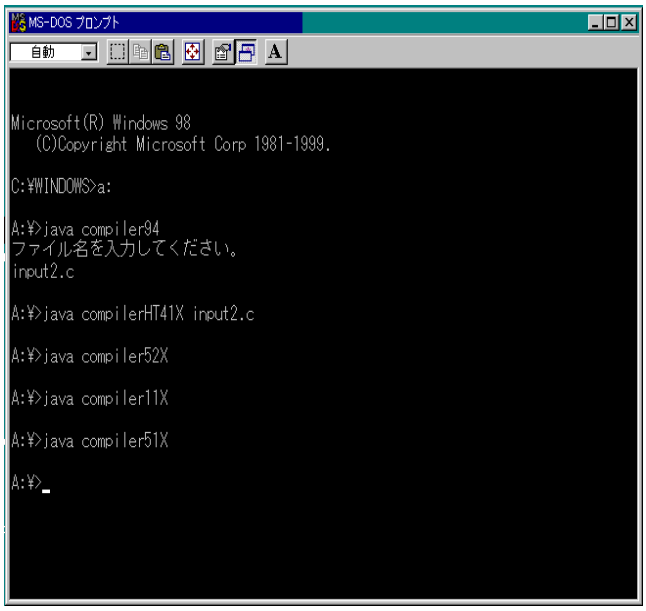


図 5-1 : 実行画面

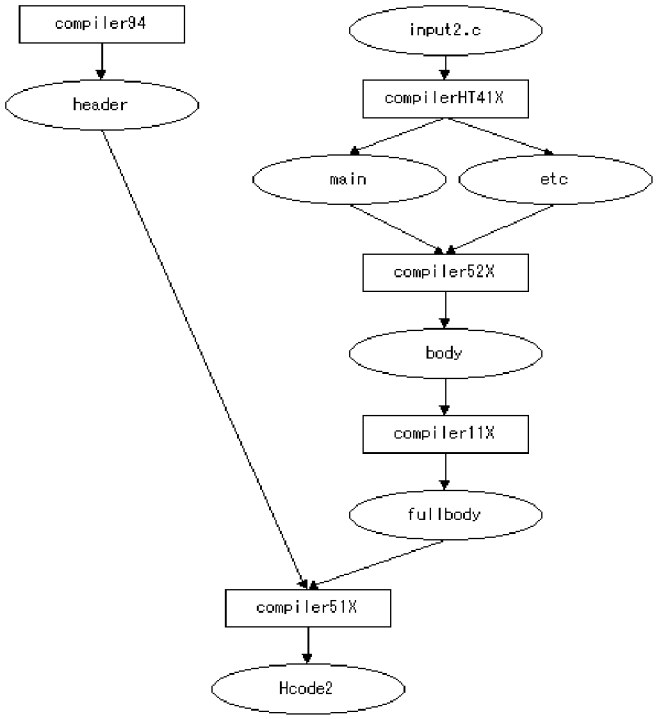


図 5-2 : 実行の流れ

6 おわりに

本研究は, C 言語のソースプログラムを Hichart 図で表示する事を目標とした. 現段階では, C 言語のソースプログラムから H-code2 を生成する Java プログラムの一部が完成した. このプログラムにより一部の条件分岐部分以外は全て C 言語のソースプログラム H-code2 への変換が可能である.

将来はさらに進化し, 文法を用いた Hichart 図描画システムや, Hichart-C トランスレーター,Hichart 図の DXL へのトランスレータなどへの統合をする必要がある. そして, プログラミング教育や個人個人のプログラム開発支援システムとして広く, また有用で使いやすいものへと発展していくことを望まれる.

参考文献

[1] Yosihiro Adati, Youzou Miyadera, Kimio Sugita, Kensei Tsuchida, Takeo Yaku, A Visual Programming Environment Based on Graph Grammarsand Tidy Fraph Drawing, ICSE98, (1998), 74-79

[2] Youzou Miyadera, Takeo Yaku, Hideaki Konya, An Expression Method for Circulation of Tree-Stuctured Diagrams 東京電機大学理工学部紀要第 19 巻, (1997), 41-59

[3] 安達由洋, 大井裕一, 大澤優, 二木厚吉, 夜久竹夫, DXL 対応 Hichart プログラム図に対する属性グラフ文法, 日本大学文理学部自然科学研究所研究紀要第 33 号, (1998), 149-164

[4] 大井裕一, Hichart を用いたプログラム開発支援システム, 東洋大学大学院工学研究科, (1995), 1-54

[5] 谷越 毅, 属性グラフ文法に基づいた構文指向 Hichart エディタの JAVA による実現, 日本大学文理学部応用数学科夜久研究室資料 (1996)

[6] 菊池潤也, Hichart プログラムエディタのコマンドに対する属性グラフ文法, 日本大学文理学部応用数学科研究室資料 (1996)

[7] 宮崎征宏, 属性グラフ文法に基づいた Hichart プログラムエディタの JAVA による実現, 日本大学文理学部応用数学科研究室資料 (1997)

[8] 大津博之, Hichart エディタの JAVA 言語による実現, 日本大学文理学部応用数学科研究室資料 (1999)

[9] 宮寺庸造, H-code2 Version0.33 の仕様

[10] 結城浩, "Java 言語プログラミングレッスン上", ソフトバンクパブリッシング社, 1999, 東京都, 354

[11] 結城浩, "Java 言語プログラミングレッスン下", ソフトバンクパブリッシング社, 1999, 東京都, 319

付録

(1)input2.c(入力ファイル)

```
#include<stdio.h>
float sankaku(int,int);
main(){
   int t,h;
   float a;
   printf("底辺・高さを入力してください。\n");
   scanf("%d %d",&t,&h);
   a=sankaku(t,h);
   printf("面積＝%f\n",a);
}

float sankaku(int x,int y){
   float ans;
   ans=x*y/2.0;
   return ans;
}
```

(2)header

```
H-code  Version 1

header{
        2000/10/7 22:6:5
        application_name Version
        unit{1 point}
        font{normal}
}

tree{
        tree_header{
                tree_name
                2000/10/7 22:6:5
                application_name Version
                {C}
                {} {} {} {}
                fontScale{6}
         }
         {predefined_process ""
         AllParentPos{center}
         AllFontGray{.2} AllCellGray{.2} AllLineGray{.4}
         cellStr{"input2.c" font{bold} fontRGB{.7 .2 .2}}
```

(3)main

```
{predefined_process"" cellStr{"main" font{bold}}

{ title var
{ name " t , h "
{ process int }
}
}
{ title var
{ name a
{ process float }
}
}
{ data_output "printf ("底辺・高さを入力してください。\n") " }
```

```
{ data_input "scanf("%d%d",&t,&h) " }
{ process " a=sankaku(t,h) " }
{ data_output "printf ("面積＝%f\n",a) " }
}
```

(4)etc

```
{ predefined_process "  #include<stdio.h> }
{ name " float sankaku ( int , int ) " }
{predefined_process"" cellStr{"sankaku" font{italic}}
{ title parameter
{ name x
{ process int }
}
}
{ title parameter
{ name y
{ process int }
}
}
{ process float}
{ title var
{ name " ans "
{ process float }
}
}
{process " ans=x*y/2.0 " }
{process " return ans " }
}
```

(5)body

```
{ predefined_process "  #include<stdio.h> }
{ name " float sankaku ( int , int ) " }
{predefined_process"" cellStr{"sankaku" font{italic}}
{ title parameter
{ name x
{ process int }
}
}
{ title parameter
```

```
{ name y
{ process int }
}
}
{ process float}
{ title var
{ name " ans "
{ process float }
}
}
{ process " ans=x*y/2.0 " }
{ process " return ans " }
}
{predefined_process"" cellStr{"main" font{bold}}

{ title var
{ name " t , h "
{ process int }
}
}
{ title var
{ name a
{ process float }
}
}
{ data_output "printf ("底辺・高さを入力してください。\n") " }
{ data_input "scanf("%d%d",&t,&h) " }
{ process " a=sankaku(t,h) " }
{ data_output "printf ("面積＝%f\n",a) " }
}
```

(6)fullbody

```
        {predefined_process " #include < stdio.h > },
        {name " float sankaku ( int , int ) " },
        {predefined_process"" cellStr { "sankaku" font { italic } }
                {title parameter
                        {name x
                                {process int }
```

```
                }
            },
            {title parameter
                {name y
                    {process int }
                }
            },
            {process float },
            {title var
                {name " ans "
                    {process float }
                }
            },
            {process " ans=x*y/2.0 " },
            {process " return ans " }
        },
        {predefined_process"" cellStr { "main" font { bold } }
            {title var
                {name " t , h "
                    {process int }
                }
            },
            {title var
                {name a
                    {process float }
                }
            },
            {data_output "printf ( "底 辺・高 さ を 入 力 し て く だ さ い 。
\n" ) " },          {data_input "scanf ( "%d%d" , &t , &h ) " },
            {process " a=sankaku ( t , h ) " },
            {data_output "printf ( "面積＝%f\n" , a ) " }
        }
```

(7)Hcode

```
H-code  Version 1

header{
        2000/10/7 22:6:5
        application_name Version
        unit{1 point}
        font{normal}
}

tree{
        tree_header{
                tree_name
                2000/10/7 22:6:5
                application_name Version
                {C}
                {} {} {} {}
                fontScale{6}
        }
        {predefined_process ""
         AllParentPos{center}
         AllFontGray{.2} AllCellGray{.2} AllLineGray{.4}
         cellStr{"input2.c" font{bold} fontRGB{.7 .2 .2}}

                {predefined_process " #include < stdio.h > },
                {name " float sankaku ( int , int ) " },
                {predefined_process"" cellStr { "sankaku" font { italic } }
                {title parameter
                                {name x
                                        {process int }
                                }
                        },
                        {title parameter
                                {name y
                                        {process int }
                                }
                        },
                        {process float },
                        {title var
```

```
                        {name " ans "
                                {process float }
                        }
                },
                {process " ans=x*y/2.0 " },
                {process " return ans " }
        },
        {predefined_process"" cellStr { "main" font { bold } }
                {title var
                        {name " t , h "
                                {process int }
                        }
                },
                {title var
                        {name a
                                {process float }
                        }
                },
                {data_output "printf ( "底辺・高さを入力してください。\n" ) " },
                {data_input "scanf ( "%d%d" , &t , &h ) " },
                {process " a=sankaku ( t , h ) " },
                {data_output "printf ( "面積＝%f\n" , a ) " }
        }
    }
}
```