山口大学教育学部附属教育実践総合センター研究紀要第13号（2002.3）

C言語プログラム解析システム

葛 崎偉[1]　嶋田 一行[2]　斗納 宏敏[2]

An Analytical System for C-Language Programs

Qi-Wei Ge[1], Kazuyuki Shimata[2] and Hirotoshi Tonou[2]

(Received November 6, 2001)

あらまし　ソフトウェア開発における課題として、生産性や品質の向上が挙げられている。その解決の手法として、既存のソフトウェアの修正、改良および追加によるアプローチが考えられる。そのとき、既存のプログラムの構成要素を分析し、ソフトウェア全体の設計および仕様を再確認する必要があり、また追加や修正の部分を含めて全体が新しい仕様に合致することを確かめる作業も必要である。そのため、プログラムの仕様や設計を容易に理解できる表現技術が必要不可欠となる。そこで本研究は、C言語で書かれたプログラムの構造を解析し、フローチャート等でプログラムそのものを視覚的に表現することによって、解析対象のプログラムの読解を手助けするシステムの開発を目的としている。C言語のプログラムは階層構造をとっており、その基本単位は関数としている。そのことから、解析は主として関数単位で行うこととする。解析結果は、関数ごとに処理の流れをフローチャートで出力し、さらに、プログラム全体の階層構造を表すために、関数間の呼び出し関係を表した関数関連図として図示する。

1. はじめに

情報技術の発展と共に、ソフトウェアの需要が益々高まってきている。ソフトウェアの開発における課題として、生産性や品質等の向上が挙げられており、それらを向上させる方法論が注目され、様々な提案が行われている [1]。

ソフトウェアの開発には、新規に開発する場合と、既存のソフトウェアの修正や改良による場合がある。既存のソフトウェアの場合には、さらに、既存のプログラムの個々の要素からソフトウェア全体までの設計や仕様を再確認する必要があり、その上で追加や修正の部分を含めた全体が正しく仕様に合致することを確かめる作業が加わる。現実には、この作業に人間の多大な思考と労力を要している。なぜなら、既存のプログラムとは、他人が書いたものや、自分が書いたものでも書いてから時間が経過しているものであるため、例えそのプログラムに細かく注釈が付いていても理解には困難を要

[1]山口大学教育学部
[2]富士通テン株式会社

するからである [2]。従って、ソフトウェアの開発において、プログラム要素の一つ一つの仕様や設計を容易に理解できる技術が必要不可欠となる。それに対応するために、CASE ツールの開発が進められている [3]。CASE ツールの開発には、システム仕様の確認が要求されており、これは開発したプログラムの構造および実行の流れが仕様に合っているかどうかをチェックするものである。これに関連して、プログラムを木構造で表現したり、プログラムをデータベースで管理する研究が行なわれている [4, 5]。

ここで問題となるのは、プログラムそのものをどう視覚的に表現するかという点である。今まで、ＰＡＤ図やＹＡＣなどの表記法の研究開発も行なわれてきたが [6]、ソフトウェア開発の現場からは未だに古くから広く使われてきたフローチャート表記の要求がある。そこで本研究では、Ｃ言語プログラム [7] を対象に、プログラムを解析し、プログラムの構造をフローチャートと関数関連図で表現することで、ユーザが被解析プログラムの処理の流れを理解するのを助けるシステムの開発を目的とする。対象とするのは、Ｃ言語で記述されたプログラムである。

Ｃ言語は、UNIX というオペレーティング・システムを記述するために作られ、UNIX と共に広く普及した。現在では、UNIX や UNIX 上のソフトウェア・ツールだけでなく、パッケージ・ソフトの記述などにも多く使用されており、MS-DOS などパソコン用の OS 上でのＣ言語の処理系も多くある [8]。これは、Ｃ言語の移植性の良さや、汎用性によるものと言える。アセンブラ言語で書かれたプログラムは互換性が少なく、プログラミングにも時間がかかるので、生産性向上のためにＣ言語で書き換えられているものも少なくない。また、ダウンサイジングにより普及してきたワークステーションは UNIX を使用しているし、近年進められているネットワーク化のためのツールもＣ言語で記述されているものが多い。このように、今後さらにＣ言語が普及していく要因は数多く、Ｃ言語による開発が増加すると思われることから、Ｃ言語を対象とした。

2. システムの概要および機能

2.1 システム概要

この解析システムで解析を行うＣ言語は関数型言語と言われ、そのプログラムの基本単位は関数と考えられている。つまりＣ言語プログラムは、いくつかの関数が複雑に繋がり合って一つのプログラムを構成していると考えられる。

そこで、各関数間の呼び出し関係を調べ図示し、さらに各関数ごとにアルゴリズムをフローチャートで図示することによって、プログラム全体を表現することが出来るのではないかと考えた。この考えに基づいて、プログラム解析のためのシステム開発を行なった。本システムの機能は、関数間の関連の解析と関数内部の解析の２つの機能を持つ。なお本システムは、開発に用いた Sun ワークステーション（OS:solaris1.1）上のＣ言語プログラムに対応し、またその後に定められた ANSI 規格に準拠したＣ言語プログラムに対応できるように開発を行った。また、Ｃ言語プログラムにおいて goto 文、longjmp 関数、setjmp 関数は制御の流れを不規則に変えるため嫌われており、今回の解析の対象外にした。

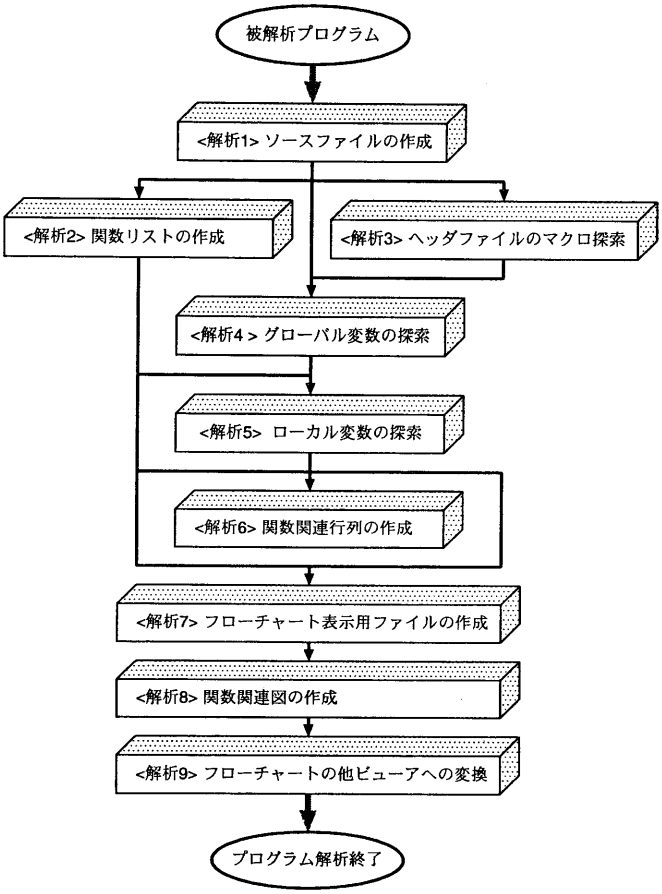

図 1 システムの機能

2.2 システム機能

本システムの機能は、Makefile というプログラムの生成法を記したファイルから、ユーザが定義したプログラムを解析し、解析結果を表示するものである。図 1 にあるように、本システムのプログラム解析機能は、関数間の呼び出し関係の解析機能と、関数内部のアルゴリズムの解析機能に分けられる。

まず最初に、Makefile からプログラムが書かれているソースファイルをすべて見つけてくる（解析 1）。そしてソースファイル毎に、その中にある関数名のリスト化（解析 2）、ヘッダファイルにあるマクロの調査（解析 3）、グローバル変数の調査（解析 4）、ローカル変数の調査（解析 5）、関数の呼び出し関係の調査（解析 6）を行い、その結果をファイルとして残す。本システムの解析機能としてあげられている 2 つの機能は、このようにしてソースファイル毎に作られたファイルを使い、関数間の呼び出し関係は解析 8 で関数関連図として図示し、関数内部のアルゴリズムは解析 7 と解析 9 でフローチャートとして図示する。

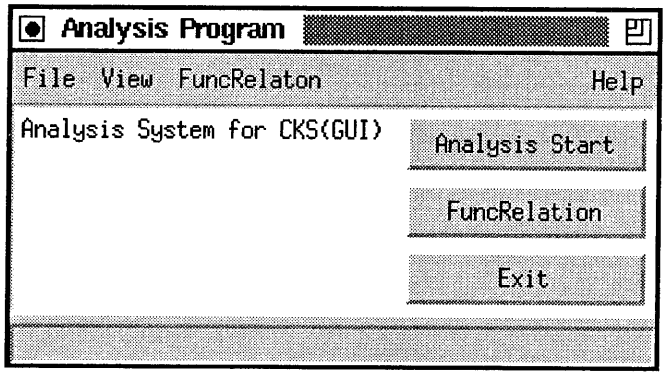

図 2 インタフェース

2.3 作成されるファイル

先に述べたように、最終的に図 1 の解析 7 、 8 、 9 で解析結果を図示するが、それ以前に行われた解析の結果はファイルに保存される。作成されたファイルは解析 7 、 8 、 9 以外にも使われている。たとえば、解析 2 で作られた関数名が保存されているファイルを利用し、解析 6 において関数間の呼び出し関係の解析が行われる。

このように一連の解析で作成されるファイルは、その用途の違いにより、以下のようになる。

- 被解析プログラムに書かれているデータ
  - ファイル名や関数名および、グローバル変数など関数の外側で定義されているもの
  - ローカル変数のように、各関数の内部で定義されているもの

2.4 結果の画面表示

本システムは、解析結果としてフローチャートと関数関連図を図示するために、Tcl/Tk[9] の機能を用いて X Window 上に、フローチャートと関数関連図を表示している。また、図 2 に示すように、この Tcl/Tk を使い、本システムのインターフェースを作成している。

このインタフェースは、マウスによる操作で、以下にあげる機能が実行される。

- 解析を行うのに、必要なファイルとデータの参照
  - プログラム解析に必要な情報を納めたファイル
  - 解析対象となるプログラムの [Makefile]
  - 本システムを動作させるのに必要な環境変数

- 関数関連図の作成と表示

これにより、本システムで解析解析の対象となるファイルと、解析結果としての関数関連図の表示が行われる。

3. フローチャート作成

本章では、フローチャート作成についての考察、フローチャートの作成結果、およびその表示結果について説明する。

3.1 フローチャート作成の考え方

関数の内部でどのようなことが行われているか、つまりアルゴリズムの図示において、そのアルゴリズムがどのような流れを持つのかがすぐ理解できることが求められている。本研究では、関数のアルゴリズムの解析を行いそれを図示するものとして、古くからアルゴリズムの表記法として使われているフローチャートを選んだ。

フローチャートは、データ記号や処理記号によって表されるデータが、それらを結ぶ線によって複雑につながっている。フローチャートを表現するには、データ記号や処理記号の一つ一つを1個のブロックとみなして考え、プログラムの内容にあわせて、それぞれのブロックの役割やブロック同士のつながりを示せばよい。そこで関数の内部を、構文の内容によってブロック単位に分割し、ブロックの役割とつながりを解析すれば、フローチャートを作成できる。

まずフローチャートを作成するには、関数をブロック単位に分割し、ブロックの役割・つながりを解析した結果のファイルを作成する。そして、このファイルにある結果を使いTclスクリプトなどを用いて画面にフローチャートを表示する。

以降の説明のため、以下の言葉を定義する。

ブロック フローチャートに用いられるデータ記号や処理記号1個分のデータを持つ。各ブロックに番号を付け、ブロックの内容、役割、つながりはすべて番号で管理する。

定義関数 被解析プログラム中で定義されている関数。

トークン C言語プログラム解釈上の、意味のある文字または文字列のこと。記号や定数も含まれる。トークンは、図3のように分けられる。

フローチャートを作成するためには、例えば、プログラム上に"for"というトークンが出現した場合、そのトークンが制御を表し、そこからループが始まるという事を認識しなければならない。そのため、関数ごとに分けられたプログラム上の一つ一つのトークンまで細かく調べる必要がある。そこで、フローチャート作成に必要な解析結果

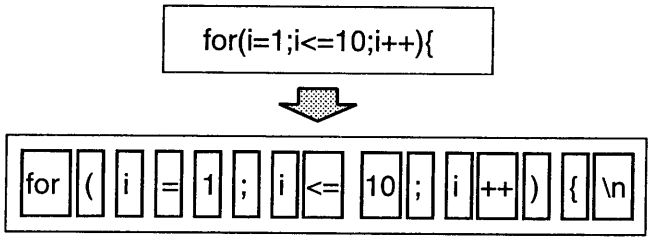

図 3 トークンとは

表 1 トークンの働き

[宣言]	[制御]	[グローバル変数]
[定義関数]	[未定義]	[その他]

だけでなく、他の機能実現に必要でトークン単位での解析を要するデータもファイルの形で保存しておくことにする。

トークンの働きは、以下の 6 種類に分類される（表 1 ）。これらの分類はブロックの役割を決める上で非常に重要となる。

3.2 ブロックの作成

関数の内部をフローチャートで表示を行うためには、関数内部をフローチャートで用いられるブロック単位に分割しないといけない。関数内で行われるブロックの作成は、図 4 のようにトークンがどのような分類かを調べていく事から始まる。

このようにして、その行がどのような分類になるかを決定し、ブロック単位でどうまとめるかを決める。その分類が制御文、定義済み処理以外で直前のブロックの分類と同じであるのなら複数のブロックに分割せず、直前のブロックの一部になる。制御文などは、そこから複数のブロックに分かれることがあるので、その行は単独で 1 つのブロックを形成する。そして、ブロックに何ブロック目かを示す番号やブロック階層（次節参照）を付けて次の行の解析に移る。これを関数の最後まで繰り返した後、分類したブロックをファイルに書き込み、別の関数でブロックの作成を行う。

3.3 ブロックのつながり

前節で関数内部をブロック単位に分割したので、次にブロックをどのようにつなげるかを調べていく。

ブロックのつながりは、基本的にはプログラムの記述における上下関係であるので、

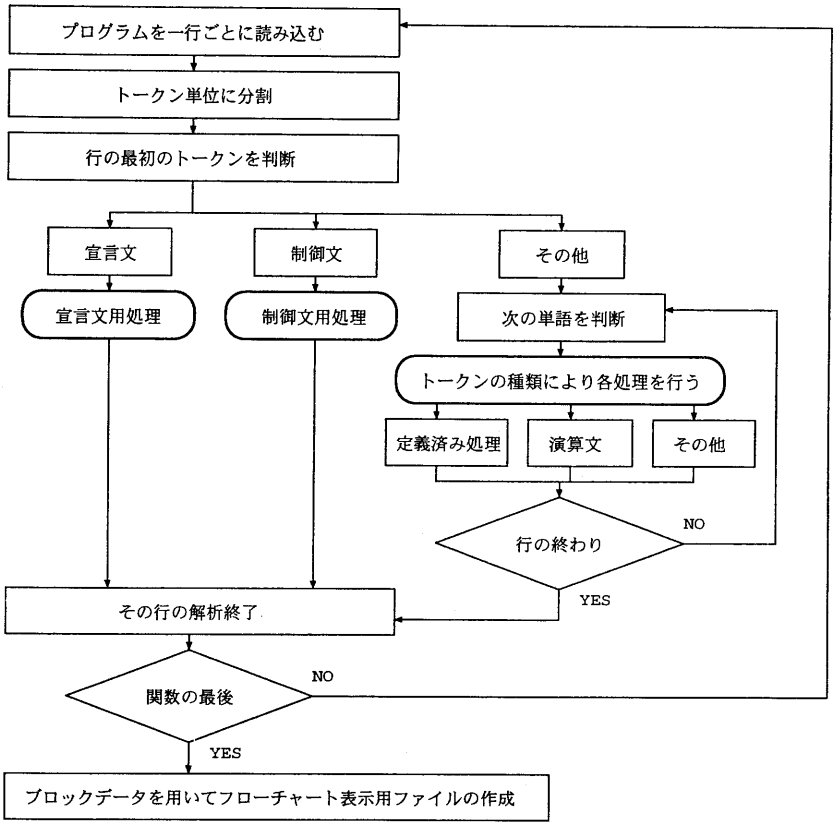

図 4 ブロックの分類

ブロックも番号通り順番につなげればよい。しかし、条件分岐のような制御文が現れると、記述における上下関係は崩されてしまう。そこで制御文が現れた場合を考える。

C 言語の制御文は、goto 文を除くと、ループ（for、 while、 do）、スキップ（break、continue、 return）、分岐（if、 switch）の 3 パターンに分けられる。スキップ処理は制御の行き先を調べ、行き先となるブロックとつなげればよいが、ループ処理と分岐処理の場合、制御の流れが複雑になっている。そこで、これらの制御文を一つのブロックとみなして考える。制御文がどことつながるかを調べ、制御文の中にある文をブロックとして表した時の最初と最後のブロックを、制御文の前後のブロックとつなげればよい。

この「ブロックの中のブロック」を表すために、ブロックの作成時にブロック階層を持たせている。このブロック階層は、そのブロックが含まれている制御文の数を表していて、階層が 1 増えるごとにその階層の制御の種類を記録させ、階層が 1 減るときがその制御の終端となる。従って、次にくるブロックの階層が同じならそのままつなげればよく、階層が違う場合はブロックの分類を調べ、分類にあわせ別のブロックとつなげることになる。

```
FUNC(a,b)
    int a,b[12];
{
  int i,j;
  i=1;
  j=10;
  for(i=1;i<=10;i++,j--){
    b[i]=i*j;
    if(b[i]<15){
      a=i+j;
      printf("%d+%d=%d \n",i,j,a);
    }
    else{
      a=i-j;
      printf("%d-%d=%d \n",i,j,a);
    }
  }
}
```

図 5 サンプルプログラム

3.4 フローチャートの表示結果

これまでに述べた手法を用いて、実際にコーディングを行い、解析結果を得ることができた。

これから一つの例として、図5 にあるプログラムを解析し、画面にフローチャートを表示した結果を紹介する。

フローチャートの表示を行うために、下にある二つの処理が行われる。

1. 解析対象となるプログラムのブロック化
2. フローチャートの作成

以後、これらについて、詳しく説明していく。

1. **解析対象となるプログラムのブロック化**

   ここでは関数毎に、ブロックの内容を示すファイル（図6）と、ブロックの内容やどのブロックとつながっているかを示すファイル（図7）の二つが作られる。

   まず、図5のプログラムを、1行毎にブロック分けを行うと、図6のように分かれる。このファイルは、アットマーク（@）から始まるブロック番号の行と、プログラムの内容が書かれている行が交互に並んでいる。ブロック番号とは、プログラムの冒頭から数えて何番目のブロックかを示し、ブロック番号の次の行に、そのブロックに当てはまるプログラムに書かれている構文が入る。この表がファイル [FlowCont] となる。

```
@0
FUNC
@1
   int a,b[12];
@2
{
@3
  int i,j;
@4
  i=1;
  j=10;
@5
i=1;i<=10;i++,j--
@6
    b[i]=i*j;
@7
b[i]<15
@8
    a=i+j;
@9
    printf("%d+%d=%d \n",i,j,a);
@10

@11

@12
    a=i-j;
@13
    printf("%d-%d=%d \n",i,j,a);
@14

@15

@16
```

図 6 [FlowCont] の内容

次に、プログラムとブロックデータのファイル [FlowCont] を用いて、各ブロックの役割とつながりを表しているファイル図 7 を作成する。図 7 の各行は順に、対応するブロック番号 : ブロックの階層 : ブロックの分類 : 接続している（次の）ブロックを表している。接続しているブロックは制御文などでは複数存在するので、データの区切りを表すフラグとして各行の最後に”−1” を用いている。この表がファイル [FlowStru] となる。

2. **フローチャートの作成**
関数毎に作られた、プログラムをブロックに分けた [FlowCont]、ブロックの種類やブロック同士のつながりを表した [FlowStru] を用いて、フローチャートを作成していく。

```
 0  0 222   1  -1
 1  1   1   2  -1
 2  1 220   3  -1
 3  1   1   4  -1
 4  1   5  -1
 5  1 207   6  -1
 6  2   4   7  -1
 7  2 201   8  11  -1
 8  3   4   9  -1
 9  3   0  10  -1
10  2 202  14  -1
11  2 205  12  -1
12  3   4  13  -1
13  3   0  14  -1
14  2 206  15  -1
15  1 208  16  -1
16  0 223  -1
```

各行の2番目の数字は以下のブロック分類を表す。
以下に対応しているブロック分類を示す。

```
  0:その他
  1:宣言文
  4:演算文
201:ifstr(if文の始まり)
202:ifend(if文の終わり)
205:elsestr(else文の始まり)
206:elseend(else文の終わり)
207:forstr(for文の始まり)
208:forend(for文の終わり)
220:tempstr(中カッコのみの制御の始まり)
222:funcstr(フローチャートの始まり)
223:funcend(フローチャートの終わり)
```

図 7 [FlowStru] の内容

まず [FlowStru] を読み込んで、ブロックの階層と分類を調べる。またブロック毎に [FlowCont] にあるブロックの内容を結びつけ、プログラムのソースファイルでそのブロックが何行目にあるかを調べる。これらの結果をブロック毎に、ブロックの分類、ブロック階層、ブロックが当てはまる行が何行目であることを示す数字、そのブロックの内容の順でファイルに保存する。こうして作られたファイル [FlowChart.c2c] 図 8 を、ビューアに読み込ませて表示すると図 9 のようになる。

```
0000:0:0:/home/apply/cks/testfile/FUNC/List
0001:0:1:FUNC
0000:1:5:  i=1;
0001:2:7:i=1;i<=10;i++,j--
0000:3:8:    b[i]=i*j;
0004:3:9:b[i]<15
0000:4:10:       a=i+j;
0000:4:11:       printf("%d+%d=%d \n",i,j,a);
0009:3:11:
0000:4:14:       a=i-j;
0000:4:15:       printf("%d-%d=%d \n",i,j,a);
0000:1:15:
0002:0:0:end_proc
```

図 8 [FlowChart.c2c] の内容

図 9 にあげたビューアは、フローチャートの表示と共に、以下にあげたことができる。

- その関数のプログラムのソースとフローチャートが並べて表示される。

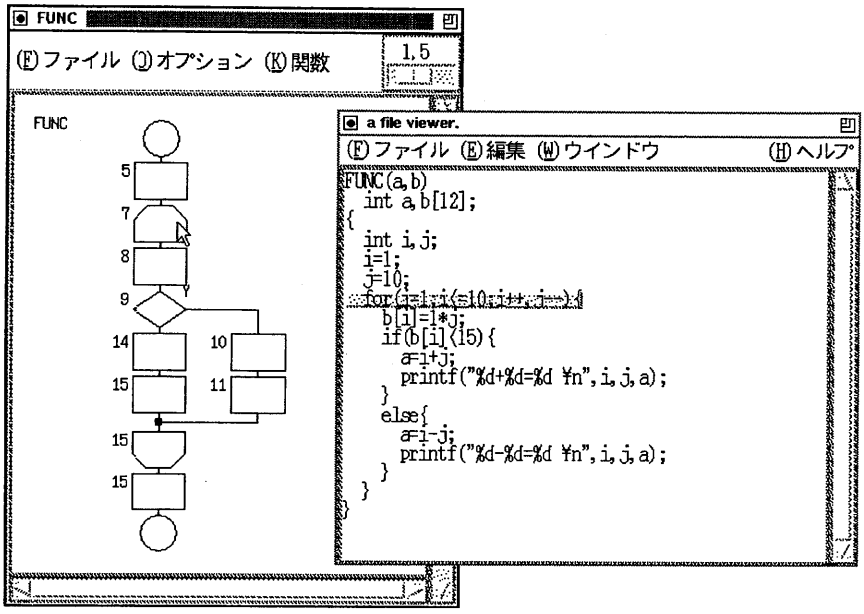

図 9 フローチャートの表示結果

- フローチャートのブロックをクリックすることで、ソースファイルの該当する部分を明示される。

これによって、フローチャートのブロックと、ソースファイルとの対応がなされ、アルゴリズムが理解しやすくなっている。

4. 関数関連図作成

本章では、プログラムの内容理解のために行われる、関数の呼び出し関係を図示した関数関連図の作成について説明する。

4.1 関数関連図の表示の準備

先に述べたように、C 言語は複数の関数からなる関数型言語である。そのため、プログラムの内容理解のために、関数の中と外に分けて図示を行う。関数の中でどのように処理が行われているかは、3 章で述べたようにフローチャートによって図示している。それとは別に、プログラムにおいて複数存在する関数がどのように使われているかを示したのが、これから説明する関数関連図である。

この関数関連図とは関数間の呼び出し関係を図に表したもので、本システムによる解析結果の 1 つとして出力される。

この関数関連図の作成のために、関数間の関連について解析を行う。プログラム中で定義してある関数をすべて調べ、関数毎に定義している関数を呼び出したかどうかを隣接行列で表す。このようにして得られた隣接行列を用いて、関数関連図を作成する。

今回はmain関数から始まる、すべての関数間の関連図を作成する機能を開発した。この機能を実現するためには、以下の手順を踏む。

1. 関数間の呼出関係（隣接行列）を表したファイルを読み込む。
2. 関数の名前を記しているファイルを読み込む。
3. 呼出関係を表した隣接行列と、関数の名前から、関連図を描く

4.2 関数関連図の作成

関数関連図の作成するためには、あらかじめ隣接行列の形として、定義してある関数の呼び出し関係を調べておく。これを元に深さ優先探索によって、関数関連図を作成していく。

関数関連図の描画は、main関数から始めていく。最初にmain関数を書き込んでから、main関数から呼び出されているすべての関数の書き込みを行うが、関数一つ一つに対してどのように関連図に書き込むかは、すでに関連図に書き込まれたかどうかを調べ書き込まれている場合と書き込まれていない場合に分かれる。

- 書き込まれている場合
  - この関数を、背景の色を変えて関連図に書き込む
- 書き込まれていない場合
  - この関数を、関連図に書き込む
  - また、この関数に隣接する関数の書き込みを行うが、このときも書き込まれている場合と書き込まれていない場合にわけて書き込みを行う

このように繰り返して、関数が書き込まれているかどうかを調べながら関連図に書き込んでいき、書き込むことができる関数がなくなるまで繰り返すと、関数関連図が完成する。

4.3 関数関連図の表示結果

実際に関数関連図の表示を行うと、次のようになる。ここでは、図１０に示すプログラムから、関数関連図を作成していく。

このプログラムは、main関数と４つの関数からなるプログラムである。このプログラムに対し、定義された関数の名前を記したファイル [DefFunc] は図１１のようにな

```
/* file1.c */

#include <stdio.h>
# ....

  ....

main()
{  ....
   func1();
   ....
   func2();
   ....
}

func1()
{  ....
   func2();
   func3();
   ....
}
```

```
/* file2.c */

#include <stdio.h>
# ....

func2()
{  ....
   func4();
   ....
}

func3()
{  ....
   func1();
   ....
}

func4()
{  ....
   ....
}
```

図 10 被解析プログラムのファイルの内容

る。このファイルには、関数に関する情報として左から、関数名、隣接行列で何番目かを示す数字、その関数があるプログラムのファイル名、という順番でデータが書かれている。

```
main:1:/.../.../file1.c
func1:2:/.../.../file1.c
func2:3:/.../.../file2.c
func3:4:/.../.../file2.c
func4:5:/.../.../file2.c
```

図 11 [DefFunc] の内容

また、関数間の呼出関係の解析により作成された、このプログラムにおける関数間の呼出関係を表した隣接行列のファイル [RelMatrix] は図１２のようになる。ここで使われている隣接行列の n 行目 m 列目は、[DefFunc] で n 番目の関数に m 番目の関数が呼び出されているかを示し、その値が 0 の場合は関数の呼出関係がなく、1 の場合は呼び出しているということになる。

[RelMatrix] と [DefFunc] から関連図を作成すると、関数関連図として図１３が作られる。四角の中にある名前が関数の名前であり、関数 A で関数 B を呼び出している場合、A から B に向けて有向枝を置く。

```
5
0 1 1 0 0
0 0 1 1 0
0 0 0 0 1
0 1 0 0 0
0 0 0 0 0
```

図 12 [RelMatrix] の内容

図１３において、灰色の背景で表示されている関数は、関数の呼び出し関係において、関数呼び出しの流れで初めて現れた関数である。白色の背景で表示されている関数は、既に他の関数から呼び出されている関数であり、別の場所で灰色の背景で表示されている。

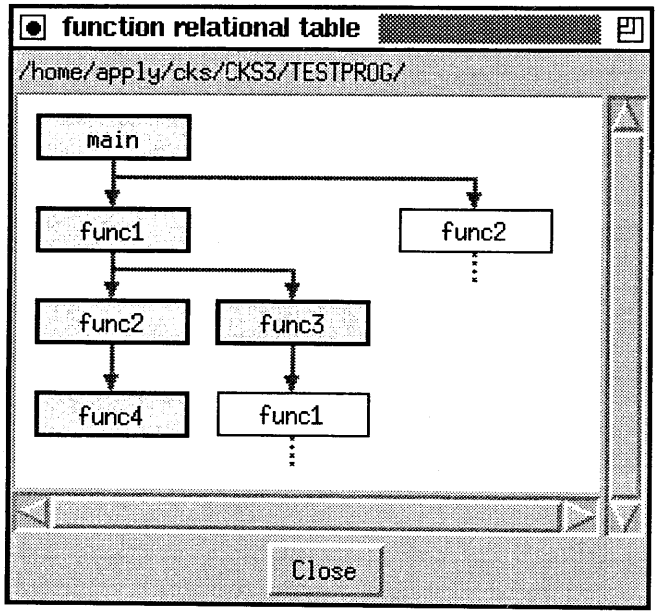

図 13 表示内容

この関数関連図には、理解しやすくするため、以下のような機能が含まれている。

1. 関連図の関数の上にカーソルを持っていくと名前が別に表示される
2. 関連図の関数をクリックすると、その関数のソースを見ることができる

1は、関数の名前を表示する機能である。関数関連図にも名前を表示しているが、表示のスペースの都合があり、名前が長い場合は見えにくくなるので、別に表示されるよ

うになっている。なお、カーソルがその関数を離れると、自動的に名前の表示部分が消えるようになっている。2は、関連図から、各々の関数のソースが理解できるようにするための機能である。クリックすることで、別のウィンドウが現れ、そこに関数のソースが表示される。

5. おわりに

本研究では、プログラム視覚化を行うために、フローチャートと関数関連図を用いたプログラムの表現およびその解析手法を提案した。この手法に基づいて、我々はシステムの開発を行なっており、そのプロトタイプを完成している。今後は実用化に向けて、フローチャートや関数関連図で表示される情報を増やし、ユーザーの要求に応じた表示や機能追加などについて検討していく必要がある。

参考文献

[1] 秋山義博ほか: ソフトウェア開発支援システム、 システムと制御、 第31巻、 第9号、pp. 653–659 (1987).

[2] D. J. Robson、 K. H. Bennett、 B. J. Cornelius and M. Munro : "Approaches to Program Comprehension"、 *The Journal of Systems and Software*、Vol. 14、No. 12、 pp. 79–84 (1991).

[3] 竹下亨: CASE 概説–コンピュータ支援ソフトウェア・エンジニアリング–、共立出版株式会社 (1990).

[4] S. Paul and A. Prakash: "A Framework for Source Code Search Using Program Patterns"、 *IEEE Trans。 Software Eng.*、 Vol. 20、 No. 6、pp. 463–475 (1994).

[5] S. Paul and A. Prakash: "A Query Algebra for Program Databases"、*IEEE Trans。 Software Eng.*、 Vol. 22、 No. 3、 pp. 202–217 (1996).

[6] 有澤誠: 構造的プログラミング、 p.2152、 昭晃堂、 (1984).

[7] Samuel P. Harbison、 Guy L. Steele Jr.: C、 A Reference Manual THIRD EDITION、p.392、 Prentice Hall Software Series (1991).

[8] JTM 企画株式会社、 荒瀬遥: 初めての人のC言語入門、 p.208、 西東社 (1991).

[9] 須栗歩人: 入門 Tcl/Tk、 p.264 秀和システム (1998).