レガシーアセンブリープログラム理解支援のためのプログラム分析計算と可視化設計について

秋山義博† 水野舜†

レガシーソフトウェア保守が存続する限り，プログラム理解とそのためのプログラム設計情報（プログラム構造やコントロールフロー情報等）を抽出する作業が続く．しかし，これは，簡単ではなく，中でも，大規模で複雑なアセンブリープログラムが最もハードな問題を与えている．この論文では，そのようなプログラムの理解問題を取り上げ，(1) プログラム理解の目的，(2) プログラム理解プロセス，(3) プログラム情報の解析的記述法とプログラム分析・可視化アプローチ，(4) プロトタイプツールの概略と評価実験結果の順に述べる．特に，プログラム情報を解析的（形式的）記述する方法と実行時間軸と条件判断軸を座標系とする空間で制御フローを表す方法を提案し，これにより，正確な分析アルゴリズム設計を実現でき，表示出力も理解しやすくできることを示す．実際に，以上に基づいて，プロトタイプ：ソフトウェアマイクロスコープ（SMS）を作成し，約数千行の大規模複雑プログラムに対して実験を行い，自動的に基本的な構造情報を突き止め表示できることを確認した．また，プログラム分析から理解までに要する操作量を，プログラムエディタ利用の場合に比べて $\sim 10^{-3}$ のオーダーで減ずることができることが分かった．

On Design of Program Analysis and Visualization for Legacy Assembly Program Understanding

YOSHIHIRO AKIYAMA† and SHUN MIZUNO†

Legacy Software Maintenance demands never-ending program understanding. The task has become difficult because the information of program structure and behaviors is sometimes not visible to human eyes. Such situation may be resulted maximized in the case of large complex legacy assembly code. After defining the objective of the program understanding task, this paper describes first a general model of the program understanding processes which may vary from programs to programs. Then, a new description method that uses a bracket notation is introduced to formulate assembly program information. The bracket notation is also used to describe the program analysis computation (algorithm) so that how all program information needed for easier program understanding is derived or transformed is identified and specified accurately. To present easily-understandable flow graphs, compared to currently proposed versions, a new way of drawing program graphs is proposed by introducing a space-time metric (axises) for the node layouting. A prototype called Software MicroScope (SMS) was developed based on these ideas and experimented to complex assembly programs of a few thousand lines. The evaluation has shown 1) automated analysis and visualization of program information, and 2) decreased workload of the program understanding by order of $\sim 10^{-3}$. The automated and accurate analysis of program information and the metric based software flow visualization were the keys for the improvement.

1. はじめに

レガシープログラムは，日々使用され，業務遂行上重要な役目をしている．しかし，長期間にわたり頻繁に変更されているために，目に見える構造と機能構造が一致しないことが起こって，複雑で理解が困難である，設計文書を作成することが楽でない，テストが楽でない等の良くない現象を示すことが多い．特に他人が書き保守されて複雑になった大規模アセンブリープログラムの理解はやさしくない．

この論文では，そのようなプログラムの理解問題を考える．まず，一般的なプログラム理解のプロセスモデルを与え，次に，プログラム情報の解析的記述と分析アプローチを述べる．プログラム可視化（＋ツールユーザインタフェース）方法では，制御グラフの表現空間を導入し，その空間内でのノード配置方法を述べる．ここで，解析的とは，必要なプログラム情報を得

† 金沢工業大学人間・情報・経営系
Human Information, Management and Computer Engineering, Kanazawa Institute of Technology

たいとき，まずそれを書き下し（表現し），次に，段階的に詳細化したり，まとめたり，さらに，陽には現れていないプログラム情報を既知情報から計算により導出する等を簡単（分析的）に行えることを意味する．これから，プログラム分析アルゴリズムの設計を正確に表すことができると考えられる．

以下の議論では，次の3つの要件を考慮する．

- 構造化，非構造化プログラムを問わず，大きい（10^4 行くらい）プログラムの**自動分析と表示**ができる．
- 表示は，見やすく理解しやすい．
- 利用者の操作量を，プログラムエディタを利用する場合に比べて$\sim 10^{-3}$ のオーダーで減らす．

プログラム分析を自動化する理由は，ユーザが，支援ツールを用いて，対話的に，分析と表示作業を適切な時間以内に完了することが難しいからである．また，見やすさと理解容易度については，後章で詳しく述べるが，情報探し作業が簡単で，意味を汲み取ることも容易であることを意味する．

実際に，IBM S/370 アセンブリープログラムについてプロトタイプツール/ソフトウェアマイクロスコープ（SMS）を作成し，数千から1万行のプログラムについて実験し，プログラム分析から情報表示までのオペレーションの量を，通常エディタを用いる場合に比べて，$\sim 10^{-3}$ のオーダー以下に減らせることを検証した．この論文では，IBM S/370 アセンブリー言語特有の述語については，その定義や詳細な説明は文献9)~11) を参照し，断わりなしに使用する．

2. プログラム理解

プログラム理解の目的を，フォワードエンジニアリングとリバースエンジニアリングの観点を踏まえて，次のように定義する．

(1) ソースプログラムに表現されている論理的構造を明らかにし，そこに実現されているアプリケーション機能を理解する（リバースプログラム理解）．

(2) アプリケーション機能からプログラム機能構造範囲を同定し実現を理解する（フォワードプログラム理解）．

プログラム理解のプロセスは，一般に，第1と第2の両方の目的を組み合わせたプロセスであり，一通りには決められない．そこで，**図1**に示すような4層から成るプロセスモデルを考え，個々の理解プロセスでは，自由にこの4層を渡り歩けるようにするのがよい．

第1層： プログラムの物理レベル情報の同定（ソース文や機械語命令をプログラミング言語や機械語形式に沿って分析）を行う．

第2層： プログラム全体の制御フロー，機能呼出関連図，データブロック構造等を把握する．

第3層： プログラムの基本機能構造の把握と理解を行う．

第4層： アプリケーション機能の所在場所と範囲を制御フロー図で把握する，また，逆に，制御フローから参照機能同定を行う．

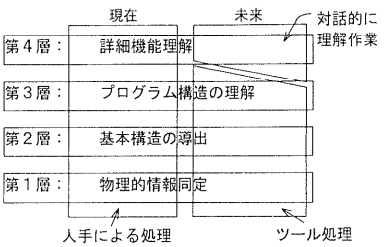

図1 プログラム理解プロセスモデル
Fig. 1 Program understanding process model.

このモデルでは，第4，3，2層レベル理解を行う場合，それぞれ，第3，2，1層レベル理解を必要とし，また，第1層レベル理解に対しては，

アセンブリー言語定義，
機械語命令定義，　(1)
マクロとプログラミングルール

が必要になる．

レガシープログラム理解支援のためには，第1~3層の作業をソフトウェアツールで**自動処理**することがきわめて望ましい．もし，プログラマが何らかの作業をする必要がある場合は，プログラムの局所部分の理解程度のものでなければならない．これを満足しない場合は，そのアプローチ自体が矛盾を含む．

3. プログラム情報モデル

プログラム理解が第1層から第4層へ進むに従って，プログラムの物理情報から全体振舞い・抽象・詳細構造等の情報の把握・理解へと進められるように工夫を考えなければならない．このために，4層に共通なプログラム情報モデルを設け，プロセスモデルを通して，順次明らかにするアプローチが効率が良い．

この共通プログラム情報モデルを見つける方法は，個々のプログラムの構造と命令文の意味を抽象化して求める方法と，コンパイラ（アセンブラ）やリンカ等が利用する**共通**情報に従う方法があるが，ここでは，後者の考えを試み，次の形式で表す．

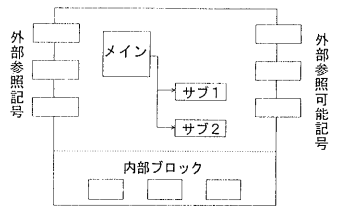

図2 プログラム情報モデル
Fig. 2 Program information model.

プログラム情報モデル

$$= \{\mathcal{N}, \mathcal{X}, \mathcal{Y}, \mathcal{E}, \mathcal{D}, \mathcal{P}, \mathcal{C}, \mathcal{G}, \mathcal{S}, \mathcal{M}, \mathcal{I}, \mathcal{W}\} \quad (2)$$

右辺記号がそれぞれ，$\mathcal{N}$：プログラムを表す名前，$\mathcal{X}$：外部参照可能記号名，$\mathcal{Y}$：外部参照記号名，$\mathcal{E}$：制御セクション名，$\mathcal{D}$：ダミーセクション名，$\mathcal{P}$：サブプログラム名，$\mathcal{C}$：プログラムコントロールフロー，$\mathcal{G}$：サブプログラム呼出関係フロー，$\mathcal{S}$：命令文，$\mathcal{M}$：マクロ文，$\mathcal{I}$：機械語形式命令文，$\mathcal{W}$：Where-Usedを意味する．ここで，Where-Usedとは，アセンブリーリストに現れるクロスリファレンスである．他の記号名はほとんど自明なので説明を省略する．

$\mathcal{N}$, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{E}$ は，他のプログラムとのリンク（インタフェース）情報であり，これ以外は，内部構造情報である．これらから，プログラムのモデルの概略図を**図2**に示すように表すことができる．

4. プログラム分析アルゴリズム設計の基本方針

プログラム理解プロセスの第1〜3層に対応するプログラム分析と第4層を支援する方法を考える．

プログラム分析への入力は，エラー文を含まないアセンブリー出力リスト（または，簡単にアセンブリーリストと以下では呼ぶ）で，**図3**に例を示す．

これより，プログラム設計を一意的に導くことができる（条件マクロが解決されているため）．また，プログラムエラーの処理が不要になる，プログラムソースコードのパーシングをほとんど省略できる等が保証される．

プログラムリストの先頭の**ESDセクション**にプログラムリンク情報（プログラム名 $\mathcal{N}$，外部参照記号 $\mathcal{Y}$，外部参照可能記号 $\mathcal{X}$ 等）が現れる．その下に続くプログラムセクションの左側（**Iセクション**）に機械語表現プログラム $\mathcal{I}$（16進数表現）が示され，その右側（**Sセクション**）にはソースプログラム $\mathcal{S}$ が示されている．最後に，クロスリファレンス $\mathcal{W}$ が示される（**CRセクション**）．

しかしながら，これだけでは，プログラム分析計算を自動化できない．つまり，SやIセクション，または，CRセクションに陽に現れている記号情報が互いに不整合であったり，または，陽には現れないが重要なプログラム情報も存在する．実際に，機械語やマクロ命令に含まれるリテラルビットデータ，オペランド内レジスタ，マクロ操作コード等すべてプログラム理解の必要情報であるが，アセンブリーリストに，つねには陽には現れない．

したがって，この不足を補うべく，以下で述べるような**命令定義基本テーブル**（3種類のテーブルから成る）（**図4**）を用意し，ユーザが，命令の種類ごとに属性の意味を指定できるインタフェースを設ける．これで，分析アルゴリズムは，入力プログラムのすべての命令文を解釈できる．

機械語命令記述テーブル（MI記述テーブル）：命令操作コード（Op）表現は記号と16進数の両方を与える．また，命令タイプ（Type），命令のレジスタフィールド位置（Ref，偶奇対，連続レジスタ列）等の情報をあわせて定義する．分岐命令は，命令操作コードが文字列“Bx”，または，16進数表現で“47”で始まるので容易に判定がつく．命令内に含まれるリテラルビットは，Iセクションの命令コードから求める．さらに，サブルーチン呼び出し用に利用するリンケージ機械語命令（BAL，BALR，命令等）やサブプログラムが他のサブプログラムに制御を渡すという意味で使用する制御非伝搬命令（BR命令等）の属性（CALLやEnd）等も指定する．

SVC命令記述テーブル（SVC記述テーブル）：監視プログラム呼び出し命令（SVC）は，そのオペランドに番号を示し呼び出す機能を指定する．しかし，実行環境が異なれば，異なる動きをすることがあるので，状況に応じてその属性を設定し，制御フロー計算ができるようにする．

マクロ命令記述テーブル（UM記述テーブル）：アセンブリー命令とマクロ命令は，統合して定義できる必要がある．たとえば，制御セクションを定義し，同時に制御フローを生成するようなマクロ文が存在するからである．実際にレガシーアセンブリープログラムでは，そのようなマクロ文が多用されている．これらの命令文の分析は，アセンブラーの文法に従う．

以上の情報を仮定して，**プログラム理解プロセスの第4層**の支援を以下のように考える．1つの機能に関連するデータ変数，マクロ/命令操作コード，リテラル，マクロパラメータ，… 等の参照を命令文ごとに調べ，全体制御フロー図上で明示することができるので，その機能が実現されている制御フロー範囲が容易

```
                                EXTERNAL SYMBOL DICTIONARY                                PAGE    1
SYMBOL   TYPE  ID  ADDR  LENGTH  LD ID FLAGS                                  ASM H V 02 17.48 07/30/89
RXCALLM   SD 0001 000000 000170        00
        RXCALLM:  Rexx CALLED function in assembler language                              PAGE    2
                                PROGRAM SECTION
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    ASM H V 02 17.48 07/30/89
                                      2 ********************************************************************** RXC00020
                                      2 * The format of the CALLED function is:
                                      3 *                                                                      RXC00030
                                      4 *       CALLED(<number><,'Detail'>,<'Asis'>)              @asis      RXC00040
                                     32 RXCALLM  CSECT                                                        RXC00320
                              00000  33          USING *,R12                                                   RXC00330
                              00000  34          USING NUCON,0                                                 RXC00340
000000 90EC D00C              0000C  35          STM   14,12,12(13)  Save the register for later.              RXC00350
000004 18B0                          36          LR    R11,R0        Get the pointer to the plist            RXC00360
                                     37          DMSFREE DWORDS=WKSZ                                           RXC00370
000006 5800 C160              00160  38+         L     0,=A(WKSZ)          SIZE OF REQUEST IN DOUBLEWORDS    01-DMSFR
00000A 0ACB                          39+         SVC   203                                                   02-DMSFR
00000C 1E04                          40+         DC    H'7684'                                               02-DMSFR
00000E 1891                          41          LR    R9,R1         Work area address                       RXC00380
                              00000  42          USING WKSECT,R9                                               RXC00390
                                     93          DMSFREE DWORDS=(0),TYPE=USER,TYPCALL=SVC                      RXC00900
000094 0ACB                          94+         SVC   203                                                   02-DMSFR
000096 1E04                          95+         DC    H'7684'                                               02-DMSFR
000098 1851                          96          LR    R5,R1         save the EVALBLOK address in R5   @asis    RXC00910
                                     98 * Fill in some of the EVALBLOK fields                        0        RXC00930
0000EE 07FE                         134          BR    R14           Return to caller                        RXC01260
                                    136 */SUB-C2B/********************************************************** RXC01280
                                    136 * C2B: Routine to co
 0000F0                             160 C2B      DS    0H                                                    RXC01520
0000F0 9002 9000              00000 161          STM   R0,R2,TEMPSAVE      Save registers                    RXC01530
00012E 07FE                         181          BR    R14                 Return                          RXC01730
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    ASM H V 02 17.48 07/30/89
000160                              204          LTORG                                                       RXC01960
000160 0000000A                     205                  =A(WKSZ)
000164 C3D4E240                     206                  =C'CMS '
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    ASM H V 02 17.48 07/30/89
000000                              213 WKSECT   DSECT                                                       RXC02010
000000                              214 TEMPSAVE DS    16F           Save area for FIND and MOVE             RXC02020
000000                              224 PARMS    DSECT               Input parameters                       RXC02120
000000                              225 NUMBER   DS    F             Relative save area slot                RXC02130
                                    3090         END                                                         RXC02410
                                CROSS REFERENCE
SYMBOL    LEN    VALUE  DEFN  REFERENCES                                   ASM H V 02 17.48 07/30/89
ASOURCE  00001 00000952 1332  1346
BEGIN    00002   00005A 0073  0084

R0       00001 00000000 0768  0036 0056 0061 0062 0066 0071 0091 0092 0103 0133 0161 0164
R1       00001 00000001 0769  0041 0056 0057 0057 0060 0061 0062 0063 0063 0066 0096 0124 0162 0172 0175 0176 0177 0178

=A(WKSZ) 00004   000180 0205  0038 0126
=C'CMS ' 00004   000164 0206  0084

                                                                          ASM H V 02 17.48 07/30/89
```

図3 プログラムリスト例
Fig. 3 An example of program listing.

に同定できる．一方，表示されたプログラム構造と機能範囲情報から，変数の参照状況/関係，“開かれたサブプログラム”の同定等も行える．

以上から分かるように，プログラム分析の最重要課題は，命令定義基本テーブルに従ってプログラムリストを自動分析し，隠れた情報を見つけて，それを陽にし情報欠落を排除することである．

5. プログラム情報記述

プログラムの構造と実行の様子を同時に記述できれば，複雑なプログラムの場合でも，分析計算の仕様を正確に記述できることになる．ここでは，便利なブラケット記法を導入し，プログラム情報の解析的表現を考える．そのためのいくつかの基本記号をまず定義する．

大文字A，B，C，…は，タイプを表す．Aを1つのタイプとするとき，$|A\rangle$ はタイプAに属する分析可能な要素の集合を表す．

裸要素は，関連情報を落とした要素で $\langle a|A\rangle_T$，または，$\langle a_A\rangle_T$ と表す．Tは，タイプAの要素の裸属性，aはその要素を代表するために必要な情報である．これは，タイプAの要素で属性Tを持つ（センシティブである）分析可能な要素aが存在するという事象を記述する．拡張として，タイプAに属する裸要素全体を $\langle|A\rangle_T$ と書く．以後，必要がない限り，裸属性T，分析可能，裸，存在する等を省略する．

要素関連情報に方向を考える必要があるとき，$|A\rangle$ はOutgoing属性付き要素集合，$\langle A|$ はIncoming属性付きタイプAの要素集合を表す．ここでは，$\langle A|a\rangle_T = \langle a|A\rangle_T$ とする．

集合 $|A\rangle$ から集合 $\langle B|$ の要素間に有向関係Tが成り立つとき，それを $|A\rangle_T\langle B|$ のように表す．この意味は，$|A\rangle$ のOutgoing属性と $\langle B|$ のIncoming属性がマッチする，あるいは，$|A\rangle$ と $\langle B|$ は接続可能であるといい，添字“T”がその有向関係を示す．これを $A \rightarrow {}_TB$ と表し，さらに，有向関係記号Tを省略するときには，$A \rightarrow B$ のように表す．

$\langle a|A\rangle_T\langle B|b\rangle$ は，要素aと要素bの間に有向関係T

```
*-------------------------------------------------------------------*
*          Machine Instruction Definition Table                     *
*                                                                   *
* MNEMONI:OP CODE SYMBOL                                            *
* TYPE:   INSTRUCTION TYPE                                          *
* LEN:    INSTRUCTION LENGTH                                        *
* COND:   CONDITION SET INSTRUCTUON(C)                              *
* PRIV:   EXECUTION MODE                                            *
*         1=PRIV., 2=PROB., 3=....                                  *
* STORAGE REFERENCE:                                                *
*         UPDT-OPRND-NO = INDEX OF OPERAND(S) WHICH IS UPDATED      *
*         REF-OPRNDS    = MARKS OF OPERAND(S) WHICH IS REFERENCED   *
*                         (UPTO THREE OPERANDS)                     *
* B:      BRANCH TYPE STMNT                                         *
* M:      MASK FOR CONDITIONAL BRANCH                               *
* C:      CALL TYPE STMT                                            *
* E:      END TYPE STMT                                             *
* REGREF: REFERENCE IMFORMATION                                     *
* REFUP:  UPDATE IMFORMATION                                        *
*         R = REGISTER                                              *
*         X = STORAGE                                               *
*         I = IMMEDIATE                                             *
*         A = ADDRESS                                               *
*         P = PAIRED REGISTERS                                      *
*         W = SCOPE REGISTERS                                       *
*-------------------------------------------------------------------*
*MNEMONI   TYPE COND PRIV OP-C STORAGE REF    M  C E  REGREF REGUP *
*           |    LEN          |   UPDT-OPRND-NO                  |  *
*           |                 |     REF-OPNDS                    |  *
*           v                 v       B                          v  *
*-------------------------------------------------------------------*
*  5   10   15   20   25   30   35   40   45   50   55   60        *
*-------|---||---|---|---|---|---|||---------|-|-------------------*
*****  General Instruction  ********************************
AR        RR  2C    2    1A   ..  ..    .    .       RR     R
A         RX  4C    2    5A   ..  .X    .    .       RX     R
AH        RX  4C    2    4A   ..  .X    .    .       RX     R
ALR       RR  2C    2    1E   ..  ..    .    .       RR     R
AL        RX  4C    2    5E   ..  .X    .    .       RX     R
NR        RR  2C    2    14   ..  ..    .    .       RR     R
N         RX  4C    2    54   ..  .X    .    .       RX     R
   .....................................
```

(1) 機械語命令記述テーブル（MI 記述テーブル）

(1) Machine instruction description table.

```
*****************************************************************
*   SVC Instruction Definition Table                            *
*****************************************************************
*     Definitions for each Column.
*          1    :  The SVC ends control flow.
*          2    :  The SVC changes Condition Code.
*          3    :  The SVC is eliminated.
*          4    :  The SVC calls a routine,
*                       the name not given explicitly.
*          Oprnd :  Operand Parms.
*
*Op.Desc.....1.2.3.4.5.6.7.8.Descriptions.....................
SVC  13       X.......
     ..........................................
```

(2) SVC 命令記述テーブル（SVC 記述テーブル）

(2) SVC instruction description table.

```
*****************************************************************
*   Macro Definition Table                                      *
*****************************************************************
* Notes:  Oprnd (Operands) must be Specified when necessary.    *
* Definition for each Column:                                   *
*   EC(15) :  The macro ends control flow.                      *
*   DA(17) :  The macro defines data area and is not executable.*
*   SC(21) :  The operand shows a call target (subroutine-ID)   *
*                1) XXXCAL    sub          (single operand)     *
*                2) XXXCAL    sub,p1=xxx,.. (First positinal word)*
*   LD(23) :  The macro defines a single DSECT logically.       *
*              ( assumes a label to be associaited)             *
*   LC(25) :  The macro defines a single CSECT logically.       *
*              ( assumes a label to be associaited)             *
*   XF(29) :  The macro is discarded and the expansion is taken.*
*    Oprnd :  Operand Parms                                     *
*****************************************************************
*          E D  S L L  X
*          C A  C D C  F
*Op.....0123456789ABCDEFGHIJ.Oprnd.....................................*
*    SECTION1: Assembler statements                             *
CSECT     . . . . . X . .
DC        . . . X . . . . .
USING     . . . . . .
*-------------------------------------------------------------------*
*    SECTION2: Structured Macros                                *
IF        . . . . . . . . X
THEN      . . . . . . . . X *
ELSE      . . . . . . . . X *
ENDIF     . . . . . . . . X *
*-------------------------------------------------------------------*
*    SECTION3: Conventional OS macros                           *
CALL      . . . . . X . . . .
RETURN    . . X . . . . . .
ABEND     . . X . . . . . .
   ..........................................
```

(3) マクロ記述テーブル（UM 記述テーブル）

(3) Macro description table.

図 4 命令記述基本テーブル

Fig. 4 Basic table of statement description.

が存在するという事象を表す．タイプ A，B に属する分析可能な要素集合 p_A，q_B がそれぞれ決定できる場合，それらを陽に示して，$\langle a|p_A\rangle_T\langle q_B|b\rangle$ と書き，$\langle a|p_A\rangle_T$ は，有効関係 T について，分析可能なタイプ p_A の要素 a 情報に対するセンシティビティを表す．$_T\langle q_B|b\rangle = \langle b|q_B\rangle_T$ も同様である．

3 つのタイプ A，B，C について，$A \to {}_{T_1}B \to {}_{T_2}C$ が成り立つとき，この関係は $|A\rangle_{T_1}\langle B||B\rangle_{T_2}\langle C|$，または $|A\rangle_{T_1}|\langle B|B\rangle_{T_2}\langle C|$ と書かれるが，これが，A と C の関係 T に同等になるとき

$$|A\rangle_T\langle C| = |A\rangle_{T_1}\langle B|B\rangle_{T_2}\langle C| \tag{3}$$

が成り立つ．これは，B を介して，それぞれ A と B，B と C の関係 T_1 と T_2 を合成するにより，A と C の関係 T になることを示し，$A \to {}_TC$ を表す．また，要素 a，c を陽に書きたいとき，次式になる．

$$\langle a_A|c_C\rangle_T = \langle a_A|A\rangle_{T_1}\langle B|B\rangle_{T_2}\langle C|c_C\rangle \tag{4}$$

タイプ A，B がをそれぞれサブタイプ $A_1, A_2, \cdots$，$B_1, B_2, \cdots$ に分解できるとき，次のように表す．

$$\begin{aligned}|A\rangle &= |A_1, A_2, \cdots\rangle \\ &= |A_1\rangle|A_2\rangle\cdots \\ |B\rangle &= |B_1, B_2, \cdots\rangle \\ &= |B_1\rangle|B_2\rangle\cdots\end{aligned} \tag{5}$$

また，A，B がそれぞれ同数のサブタイプ $A_1, A_2, \cdots$，$B_1, B_2, \cdots$ に分解できて，かつ次式が成り立つとき，

$$|A\rangle_T\langle B| = \{|A_1\rangle_T\langle B_1|\}\{|A_2\rangle_T\langle B_2|\}\cdots \tag{6}$$

$|A\rangle_T\langle B|$ は $|A_i\rangle_T\langle B_i|$ $(i = 1, 2, \cdots)$ に分解可能であるといい，$\{\}\cdots\{\}$ はそれぞれの $\{\}$ 内の接続関係の和を表す．さらに，同一の関係について，2 つの可能性が後置，あるいは前置される場合，次が成り立つ．

$$\begin{aligned}|A\rangle_T\langle B| + |A\rangle_T\langle C| &= |A\rangle_T\{\langle B| + \langle C|\} \\ |A\rangle_T\langle C| + |B\rangle_T\langle C| &= \{|A\rangle + |B\rangle\}_T\langle C|\end{aligned} \tag{7}$$

ここで，+記号は和集合をとることを意味する．

例を示す．X が外部参照可能記号タイプであるとき，$\mathcal{X} = \langle|X\rangle$ はそれらの記号集合を表し，$\langle z|X\rangle = \langle z_x|$ は $\mathcal{X}$ の 1 つの要素 z を表す．命令文番号タイプを N，命令文タイプを S とするとき，この順序対関係 NS を用いて，分析可能な番号付きソース文集合を $|N\rangle_{NS}\langle S|$ と表し，要素は，次式で与えられる．

$$\begin{aligned}&\langle \text{命令分番目} = n| \text{命令文} = s\rangle_{NS} \\ &\quad = \langle n|N\rangle_{NS}\langle S|s\rangle\end{aligned} \tag{8}$$

同様に，n 番目の命令文の次に m 番目の命令文が制御フロー（CF）として続くことを次のように表す．

$$\begin{aligned}&\langle n \text{ 番目の命令 } |m \text{ 番目の命令}\rangle_{CF} \\ &\quad = \langle n|N\rangle_{CF}\langle N|m\rangle \\ &\quad = \langle n|N\rangle_{NS}\langle S|S\rangle_{CF}\langle S|S\rangle_{SN}\langle N|m\rangle\end{aligned} \tag{9}$$

第 2 行は，**分析可能な命令文番号**の間の制御フロー

関係を表す．第3行は，命令文ノード間の制御フロー関係 $|\mathrm{S}\rangle_{\mathrm{CF}}\langle\mathrm{S}|$ を用いて“展開”した表現を与え，命令文番号とあわせて，次のように解釈する．命令文 n（$\langle n|\mathrm{N}\rangle_{\mathrm{NS}}\langle\mathrm{S}|$）に命令 m（$|\mathrm{S}\rangle_{\mathrm{SN}}\langle\mathrm{N}|m\rangle$）がコントロールフローの関係（$|\mathrm{S}\rangle_{\mathrm{CF}}\langle\mathrm{S}|$）で接続される．

同様に，$\langle|\mathrm{V}\rangle$ をプログラム中で陽に使用されている変数記号の集合，$\langle|\mathrm{W}\rangle$をその記号と参照命令番号リスト集合の対の集合とするとき，$\mathcal{W}=\langle|\mathrm{V}\rangle_{\mathrm{CR}}\langle\mathrm{W}|\rangle$は，クロスリファレンスを定義する．1つの記号 v の参照命令番号リストが w であるとき，クロスリファレンスの要素は，$\langle \mathrm{v_V}|\mathrm{w_W}\rangle_{\mathrm{CR}}=\langle\mathrm{v}|\mathrm{V}\rangle_{\mathrm{CR}}\langle\mathrm{W}|\mathrm{w}\rangle$ ように得られる．

このように，プログラムリストには，単なる番号と見える要素にも陽には表現されていない情報が付いている．これが，分析可能な命令文番号と裸の命令文番号のように，分析可能と裸を区別する理由である．

$|\mathrm{N}\rangle_{\mathrm{CF}}\langle\mathrm{N}|$ や $|\mathrm{S}\rangle_{\mathrm{CF}}\langle\mathrm{S}|$ の実現は，1通りではない．プログラムリストを直接参照する方法，事前に，2つの命令文番号についての CF 接続テーブルを作成し利用する方法，有向グラフの形式を利用する方法等，利用目的に応じた選択を行わなければならない．しかしながら，これらの実現方式は，この論文の範囲外なので，これ以上議論しない．

6. プログラム理解に必要な情報の生成可能性

プログラムリストに含まれるプログラム情報は，以下の3グループに分けられる．

(1) 完全で明示的な情報
(2) 不完全で明示的な情報
(3) 非明示的情報

6.1 完全で明示的な情報

ソースコードと他の関連セクションに陽に表れて，そこで完全に記述されているセルフデスクリプティブ情報である．プログラム名 $\mathcal{N}$，外部参照可能記号 $\mathcal{X}$，外部参照記号 $\mathcal{Y}$，制御セクション名 $\mathcal{E}$，データ構造ブロック名 $\mathcal{D}$ は記号の集合であり，$\mathcal{N}$ を除いて，$\langle|\mathrm{X}\rangle$，$\langle|\mathrm{Y}\rangle$，$\langle|\mathrm{E}\rangle$，$\langle|\mathrm{D}\rangle$ と，それぞれを表すことにする．これらは，ESD セクションから得られる．

ソース文 $\mathcal{S}$ は，(命令文番号，命令文) 順序対なので $\langle|\mathrm{N}\rangle_{\mathrm{NS}}\langle\mathrm{S}|\rangle$ と表す．N，W，S，M，I を，それぞれ命令番号タイプ，命令番号リストタイプ，命令文，マクロ文，機械語命令タイプであるとき，マクロ文 $\mathcal{M}$，機械語コード $\mathcal{I}$，クロスリファレンス $\mathcal{W}$ も，$\langle|\mathrm{N}\rangle_{\mathrm{NM}}\langle\mathrm{M}|\rangle$，$\langle|\mathrm{N}\rangle_{\mathrm{NI}}\langle\mathrm{I}|\rangle$，$\langle|\mathrm{V}\rangle_{\mathrm{CR}}\langle\mathrm{W}|\rangle$ と表すことができる．

6.2 不完全で明示的な情報

1つのブロックに明示的に現れるが，関係する他のブロックに現れない情報で，何らかの分析を行えば，完全な情報に修復できる情報である．例として，TM 命令がある．これは，

TM t, mask

の命令形式を持つ．変数 t と mask は，プログラム情報としては，同等に取り扱われなければならない．しかし，変数 t は命令コード外のメモリーを割り当てられて，かつクロスレファアレンスに現れる．mask はリテラルであり，命令コード内にメモリーを割り当てられてかつクロスレファアレンスに現れない．

このリテラルは，次のように，TM 命令文を分解して求め，クロスリファレンスに含めることができる．

$$\begin{aligned}&n\text{ 番目の命令が TM 操作コードを持ち}\\&\text{リテラル 1 をとっている事象}\\&\quad=\langle n\text{ 番目命令文, “TM” 操作コード | リテラル 1}\rangle_{\mathrm{NL}}\\&\quad=\langle n,\text{“TM”}|\mathrm{N},\mathrm{O}\rangle_{\text{NO-S}}\langle\mathrm{S}|\mathrm{S}\rangle_{\mathrm{SL}}\langle\mathrm{L}|\mathrm{l}\rangle\end{aligned}\tag{10}$$

ここで，関係 NO-S と SL は，それぞれ，(命令番号タイプと操作コードタイプの組みと命令文タイプとの間の)，(命令文タイプとリテラルタイプとの間の) 関係を表し，命令番号 n と命令操作コードが “TM” であるような命令 $\langle n,\text{“TM”}|\mathrm{N},\mathrm{O}\rangle_{\mathrm{NOS}}\langle\mathrm{S}|$ がリテラル $|\mathrm{S}\rangle_{\mathrm{SL}}\langle\mathrm{L}|\mathrm{l}\rangle$ を与える事象を記述する．

この表現から，“TM”，O，S について省略して $\langle\mathrm{L}|\mathrm{l}\rangle=\langle\mathrm{l}|\mathrm{L}\rangle$ についてまとめると，クロスレファアレンス要素 $\langle\mathrm{l}|\mathrm{L}\rangle_{\mathrm{LN}}\langle\mathrm{N}|n\rangle$ が得られる．他の参照番号を求めてリスト $\langle\mathrm{l}|\mathrm{V}\rangle_{\mathrm{CR}}\langle\mathrm{W}|\mathrm{w_l}\rangle$ にまとめることができるので，これをクロスリファレンスに追加する．このような計算を行って，不完全情報を完全情報に変換する．

サブプログラム名情報 $\mathcal{P}$ は，アセンブリープログラムの場合，プログラミングルールに従って定義される．たとえば，“BAL” 命令は，

BAL reg, label

の形式を持つ．これは，“reg に次の命令のアドレスを設定して，label で指し示す命令に制御を渡す” 命令文である．この命令文は，また，“label がサブプログラムの開始点，reg を，そのサブルーチンから戻るレジスタである ” と解釈して利用でき（式 (1) の情報から），これより次が導かれる．

$$\begin{aligned}&\langle\text{命令文番号 }n|\text{ サブプログラム label を呼ぶ}\rangle_{\mathrm{BAL}}\\&\quad=\langle n|\mathrm{N}\rangle_{\mathrm{NS}}\langle\mathrm{S}|\mathrm{S}\rangle_{\text{S-BAL}}\langle\mathrm{BAL}|\\&\qquad\times|\mathrm{BAL}\rangle_{\text{BAL-2ndOP}}\langle\mathrm{2ndOP}|\mathrm{label}\rangle\end{aligned}\tag{11}$$

したがって，命令 BAL について，命令文番号とサブプログラム名の関係を次のように定義すればよい．

$$|\mathrm{N}\rangle_{\mathrm{NP}}\langle \mathrm{P}|_{\mathrm{BAL}} = |\mathrm{N}\rangle_{\mathrm{NS}}\langle \mathrm{S}|\mathrm{S}\rangle_{\text{S-BAL}}\langle \mathrm{BAL}| \times |\mathrm{BAL}\rangle_{\text{BAL-2ndOP}}\langle \mathrm{2ndOP}| \quad (12)$$

添え字は，"BAL" 命令に関する計算を示す．これ以外のサブプログラム呼び出しのマクロ文（CALL）等についても同様の計算ができて，式 (7) を用いてそれらを合計すると，次式を得る．

$$|\mathrm{N}\rangle_{\mathrm{NP}}\langle \mathrm{P}| = |\mathrm{N}\rangle_{\mathrm{NP}}\{\langle \mathrm{P}|_{\mathrm{BAL}} + \langle \mathrm{P}|_{\mathrm{CALL}} + \cdots\} \quad (13)$$

式 (12) と式 (13) が，サブプログラム名を求める計算（アルゴリズム）を表す．

6.3 非明示的情報

プログラムリストのすべてのセクションに陽に表れない情報が存在する．たとえば，特定の機械語命令は，偶奇連続レジスタ対や多重連続レジスタを使用するとき，それらの代表レジスタだけを明記する[9]．したがって，残りのレジスタは，16 進表示の命令を分析し，以下のように求めなければならない．

n 番目命令文でレジスタを使用している事象集合

$$= \sum_{\mathrm{r}} \langle n \text{ 番目命令文} \mid \text{使用レジスタ r}\rangle_{\mathrm{NR}} = \sum_{\mathrm{r}} \langle n|\mathrm{N}\rangle_{\mathrm{NI}}\langle \mathrm{I}|\mathrm{I}\rangle_{\mathrm{IR}}\langle \mathrm{R}|\mathrm{r}\rangle \quad (14)$$

ここで，有向関係 NI，IR は，それぞれ（命令番号，機械語命令），（機械語命令，レジスタ）であり，$\sum_{\mathrm{r}}$ はレジスタ番号について和をとることを示す．関係 IR の分析を行い，命令内に隠れたレジスタ r を求める．

最後に，外部サブプログラムや OS サービスプログラムによる参照情報は，プログラムリストにはまったく現れない．しかしながら，もし，このような情報が知られている場合には，外部プログラムをコールする命令文の定義に，その参照情報として含めることができる．この場合，コール文ごとに参照情報を付加することが最も正確なので，次式で与えるのがよい．

$$\langle n \text{ 番目命令} \mid \text{使用変数 v}\rangle_{\mathrm{FV}} = \langle n|\mathrm{N}\rangle_{\mathrm{FS}}\langle \mathrm{S}|\mathrm{S}\rangle_{\mathrm{SV}}\langle \mathrm{V}|\mathrm{v}\rangle \quad (15)$$

ここで，$|\mathrm{N}\rangle_{\mathrm{FS}}\langle \mathrm{S}|$ は（外部プログラムをコールする命令文番号，コール文），$|\mathrm{S}\rangle_{\mathrm{SV}}\langle \mathrm{V}|$ は（コール文，使用変数）をそれぞれ意味し，タイプ FV は外部プログラムをコールする命令文番号→変数を示す．

7. 制御フロー計算

制御フローは，実行順に命令文ノードを接続して得られるグラフで，有向グラフによって表される．ここでは，制御フローの解析的記述に関心があるので，ブラケット記法を用いて記述することを考える．

N を命令文番号タイプとする．まず，実行可能命令文をすべて拾い，隣接する 2 つの命令文ノード間の制御フロー関係を $|\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|$ の形式で得る．これは，さらに $|\mathrm{N}\rangle_{\mathrm{NS}}\langle \mathrm{S}|\mathrm{S}\rangle_{\mathrm{CF}}\langle \mathrm{S}|\mathrm{S}\rangle_{\mathrm{SN}}\langle \mathrm{N}|$ と展開できて，ソース文の制御フロー記述部分（$|\mathrm{S}\rangle_{\mathrm{CF}}\langle \mathrm{S}|$）を閉じこめる表現も得られる．

次に，プログラム全体の制御フローを求めることを考える．全体制御フローは，1 ノードフロー，2 ノードフロー，… というように，任意の 2 つのノード間を任意個のノード鎖を制御が渡り歩くフロー**経路**の集合として次に示すように与えられる．

$$\langle \mathrm{b1}|n1_{\mathrm{N}}\rangle_{\mathrm{CF}}\langle n2_{\mathrm{N}}|*\rangle_{\mathrm{CF}}$$
$$\langle \mathrm{b1}|n1_{\mathrm{N}}\rangle_{\mathrm{CF}}\langle n3_{\mathrm{N}}|n3_{\mathrm{N}}\rangle_{\mathrm{CF}}\langle n4_{\mathrm{N}}|*\rangle_{\mathrm{CF}}$$
$$\langle \mathrm{b1}|n1_{\mathrm{N}}\rangle_{\mathrm{CF}}\langle n3_{\mathrm{N}}|n3_{\mathrm{N}}\rangle_{\mathrm{CF}}\langle n4_{\mathrm{N}}|n4_{\mathrm{N}}\rangle_{\mathrm{CF}}\langle n5_{\mathrm{N}}|*\rangle_{\mathrm{CF}}$$
$$\cdots$$

ここで，$|ni_{\mathrm{N}}\rangle$ は，タイプ N の分析可能な命令番号要素 ni $(i = 1, 2, \cdots)$ を表し，記号 $|*\rangle_{\mathrm{CF}}$ は，次を意味する．$|\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|$ の左側要素 $\langle n|\mathrm{N}\rangle_{\mathrm{CF}}$ がリターン文等のサブプログラム制御フローを終結するノードである場合（命令定義基本テーブル情報から判断できる），この右側に来る $_{\mathrm{CF}}\langle \mathrm{N}|$ 接続命令文ノードはない．これを，空ノード $\langle *|_{\mathrm{CF}}$ と陽に書いて表し，$|*\rangle_{\mathrm{CF}}$ で指定する．このノード以降には制御フローは続かない．このような形式で，制御フロー開始ノードが b1 であるときの，1 ノードフロー（第 1 行目），2 ノードフロー（第 2 行目），… を表す．また，もし，$n2_{\mathrm{N}}$ が制御フローを終了する命令でない場合，上の第 1 式の $\langle n2_{\mathrm{N}}|*\rangle_{\mathrm{CF}}$ は無意味となり，省いてもよい．他の開始ノードについても同様にかける．

以上から，隣接するノード間の制御フロー $|\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|$ を作成すると，これを用いて，プログラムの制御フローを次式で表すことができることが分かる．

$$\text{全体フロー} = \langle| \frac{|\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|}{1 - |\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|} |*\rangle_{\mathrm{CF}} \quad (16)$$

ここで，記号 $\langle|$ は，すべての制御フロー開始ノード集合を意味する．

サブプログラムの制御フローは，この全体フローから求めることができる．1 つのサブプログラム p の制御フローの開始ノードは，式 (13) の逆関係を用いて，$\langle \mathrm{p}|\mathrm{P}\rangle_{\mathrm{PN}}\langle \mathrm{N}|$ と与えられ，これと式 (16) を接続し，次式で求められることはほとんど自明である．

サブプログラム p の制御フロー

$$= \langle \mathrm{p}|\mathrm{P}\rangle_{\mathrm{PN}}\langle \mathrm{N}|\{\frac{|\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|}{1 - |\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|}\}|*\rangle_{\mathrm{CF}} \quad (17)$$

制御セクション（CSECT），ダミーセクション（DSECT），マクロ展開についても，同じ形式になる．たとえば，制御セクションとダミーセクションのフロー

とは，次の制御セクション定義文が現れるまで，命令文を次々に物理的順にたどるフローであり，次式でそれらのセクションを得ることができる．

制御セクション e

$$= \langle \mathrm{e}|\mathrm{E}\rangle_{\mathrm{EN}}\langle \mathrm{N}|\frac{|\mathrm{N}\rangle_{\mathrm{E}}\langle \mathrm{N}|}{1-|\mathrm{N}\rangle_{\mathrm{E}}\langle \mathrm{N}|}|*\rangle_{\mathrm{E}}$$

ダミーセクション d

$$= \langle \mathrm{d}|\mathrm{D}\rangle_{\mathrm{DN}}\langle \mathrm{N}|\frac{|\mathrm{N}\rangle_{\mathrm{D}}\langle \mathrm{N}|}{1-|\mathrm{N}\rangle_{\mathrm{D}}\langle \mathrm{N}|}|*\rangle_{\mathrm{D}} \qquad (18)$$

ここで，属性 EN，E，DN，D は，それぞれ，(制御セクション名，制御セクション定義命令文番号)，制御セクションフロー，(ダミーセクション名，ダミーセクション定義命令文番号)，ダミーセクションフローを表し，$|*\rangle_{\mathrm{E}}$ と $|*\rangle_{\mathrm{D}}$ は，終端ノードである．

以上で，プログラム情報モデル (2) に必要なすべてのプログラム情報を得られることを示した．

8. データアグリゲーション

データ変数をまとめる機能と制御フローグラフ上でマーキングをまとめる機能をデータアグリゲーションと呼ぶ．以下に，その基本的な計算を述べる．

ユーザが指定する変数リストには，変数，データブロック，サブプログラム参照変数の集合等，様々な組み合わせが考えられ，これを次のように書く．

$$\mathrm{U} = \{u_1, u_2, u_3, \cdots\} \qquad (19)$$

ここで，プログラム分析は，制御セクション単位に限られるので[10),11)]，変数は制御セクション内で一意的になり，u_iのタイプ（単純変数名かデータブロック名）の判定ができる．つまり，u_i の計算 $\langle u_i\rangle_{\mathrm{DV}}$ を，式 (18) を用いて，次のように行う．

$\langle u_i\rangle_{\mathrm{DV}} = u_i$ （単純変数のとき）

$$= \sum_j \langle u_i|\mathrm{D}\rangle_{\mathrm{DN}}\langle \mathrm{N}|\frac{|\mathrm{N}\rangle_{\mathrm{D}}\langle \mathrm{N}|}{1-|\mathrm{N}\rangle_{\mathrm{D}}\langle \mathrm{N}|} \times \{|\mathrm{N}\rangle_{\mathrm{NS}}\langle \mathrm{S}|\mathrm{S}\rangle_{\mathrm{SV}}\langle \mathrm{V}|\mathrm{v}_j + |*\rangle_{\mathrm{D}}\}$$

（データブロックのとき） (20)

ここで，DV 関係は，データブロック名 → 変数を意味する．最後の {} の中身は，ダミーセクションフローを追いかけて，命令文が定義する v_j 変数を拾うか，または，制御セクションが終了するノードまでたどることを表す．以下では，式 (20) で得られる変数を**素変数**，U を**ユニットデータ**と呼ぶ．

次に，サブプログラム制御フローを表す式 (17) を用いてデータ参照を考える．u を素変数とするとき，式 (17) に現れるノードのそれぞれについて，次の 3 ケースのどれかが起こる．

1. 変数 u を参照しない．
2. 変数 u を参照する．
3. 制御フローが終了する（$|*\rangle_{\mathrm{CF}}$の意味で）．

ケース 1 のとき，制御フローに沿って次のノードを選択する．ケース 2 のとき，このノードにマーキングを行い，さらに次のノードの検査へと進む．この後半の“次のノードの検査へと進む”は，ケース 1 と同等の処理である．ケース 3 のときは，制御フロートレースを終了する．これから次の式を得ることができる．

サブプログラム p の変数 u の参照制御フロー

$$= \langle \mathrm{p}|\mathrm{P}\rangle_{\mathrm{PN}}\langle \mathrm{N}|\frac{|\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|}{1-|\mathrm{N}\rangle_{\mathrm{CF}}\langle \mathrm{N}|} \times \{|\mathrm{N}\rangle_{\mathrm{NS}}\langle \mathrm{S}|\mathrm{S}\rangle_{\mathrm{SV}}\langle \mathrm{V}|u\rangle + |*\rangle_{\mathrm{CF}}\} \qquad (21)$$

この {} 内は，サブプログラム p の制御フロー開始点よりフローをたどり，変数 u を参照するノードを見つけるか，または終端ノード文であることを意味する．

以上で，プログラムリストから命令文単位のプログラム情報，またはプログラム全体情報等をプログラム分析計算により得る方法を述べた．これで，ブラケット記法がプログラム構造情報と制御フロー等の実行情報の計算の記述に有効であることが分かる．

9. プログラムの可視化

プログラム理解プロセスの第 4 層支援について，制御フロー，サブプログラム呼び出し関係，およびデータアグリゲーションの可視化方法を述べる．**制御フロー図**は，入力プログラムの状況によらずに，フローグラフのトポロジーを表示することが望ましい．したがって，処理順序や条件選択における近さをグラフに表すことを考える．以下の議論を分かりやすくするために，1 つの例を図 **5** に示す．

制御フローグラフのノードの配置は，時間軸と選択

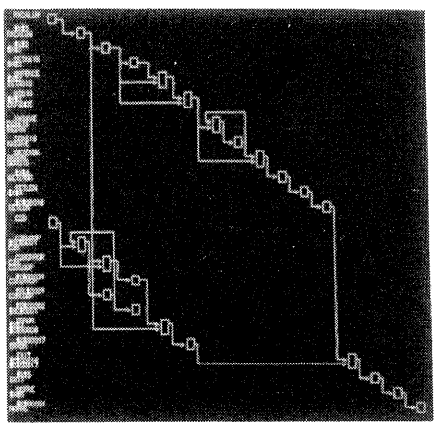

図 5 制御フローグラフ例
Fig. 5 An exapmle of control flow graph.

軸の2次元空間（時間選択空間）で表す．ディスプレイ（ウィンドウ）上の左上隅を原点とし，そこから右方向を時間軸としてノード実行順序を表し，下方向を選択軸として条件判断イベントの順序を表す．時間軸については，1命令実行時間ユニットとする．選択軸については，1回分選択とする．

時間軸方向のノード配置は，実行順に並べ一意的に決めることができる．一方，選択軸方向についてのノード配置は一意的には決まらない．これは，グラフノード配置問題を解かなければならないことを意味する．このために，制御フローグラフをアサイクリックグラフに変換しノード配置を行う通常アプローチをとり，制御フローの可視化においては，近いノードまたはクラスターどうしは近い位置に配置する「距離」概念を導入し，それを時間軸と選択軸の空間で計る．

グラフをクラスターに分割する方法を，2つのノード間のドミナント関係[19]に着目して考える．1つの有効グラフ上の1つのノードvがノードwをドミネートしているとする．さらに，この有効グラフ内の2つのノードを結ぶ有効エッジすべての向きを逆にしたグラフにおいて，ノードwがノードvをドミネートするならば，v，w対を**ドミナントノード**と呼び，そのようなノード対で挟まれた部分グラフをプロパーサブグラフと呼ぶ．これは，1入力線/1出力線を持つサブグラフである．

以上から，次に配置するノードの選択ルールを以下のように与える．

1. フロー始点ノードから終点ノードまでの経路について，より長い（ノード数の多い）経路上にあるノード
2. 配置したノードが含まれるドミナントノードが属するプロパーサブグラフ内ノード
3. ループ内ノード
4. ソースコード順ノード

この選択ルールに従えば，条件文でない命令文はノードが続く場合，単位時間，単位選択だけ**進**んだ位置にその後続ノードを配置するので，制御フロー図は，左側上部から右側下部の対角線方向に直線的に表されることが分かる．次に，1つの条件文ノードに2つのノード（“yes”と“no”に対応して）が後続する場合を考える．このときのノード配列は，次のようになる．

(1) 条件文ノードから単位時間，単位選択だけ**進**む位置に，ノード選択ルールで選択された後続ノード（これを第1被選択ノード，他を第2被選択ノードと以下では呼ぶ）を配置する．
(2) 条件文ノードの直近のドミナントノードを計算で求める．
(3) 第1または第2被選択ノードと直近のドミナントノードで挟まれた制御フロー上にあるノード配列を行う（再帰処理）．
(4) 直近のドミナントノードの配置は，ステップ3で配置したサブグラフのすべてのノードよりも右側下方に現れる．

ループエッジ線は，ループ内のすべてのノードに対してその上部を通るように配線する．

複数サブプログラムが制御フローを共有しないとき，機能的に干渉しないという．この場合の制御フローは，ソースコード番号順に配置できる．干渉がある場合，ソースコード順に初めに現れるサブプログラムの制御フローグラフの配置を行い，次に，これに最も干渉するサブプログラムのフローをその下方に配置する．

このグラフノード配置方式に従って生成される制御フロー図のいくつかの特徴を述べる．

- サブプログラムとデッドコードの開始点ノードは，時間軸の最左端に現れる．
- フロー図内のプロパーサブグラフが塊として現れ，機能領域を簡単に判別できる．
- 1つのノード以降に起こる処理フローを容易に指定できる．
- 1つのノードに影響を与える処理フローを簡単に同定できる．
- 正常処理経路がフロー線交差量が最も少ない画面上部領域に現れることが多い．

サブプログラム呼び出し関係図では，ノードがサブプログラム名を，有向エッジがコール関係を表し，横軸がサブプログラム呼び出し階層軸，縦軸がサブプログラム呼び出し選択軸である．1つのサブプログラムがコールされるとき，そのノードは，単位階層と単位選択分進んだ所に配置される．前述のグラフと同様のノード配置を行うが，選択軸に関するノード選択規則は，ここでは，省略する．この呼び出しフローが互いに干渉する（共有サブプログラムがある）場合，これも干渉するサブプログラム名をまとめて配置する．

10. データアグリゲーションの可視化

ここでは，次の支援を考える．

- 機能範囲（データ変数とその処理命令文の有機的結合）の同定と理解
- 特定変数，または変数グループについて，それらの参照関係把握

狙いは，アプリケーション機能に対応するプログラム構造と制御フロー範囲を簡単に求めること，また，

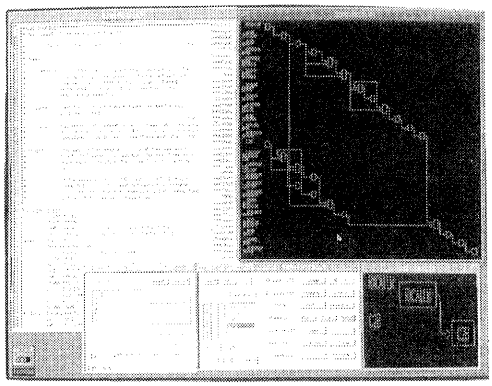

図6 プログラム情報の基本表示例
Fig. 6 An example of the basic presentation of program information.

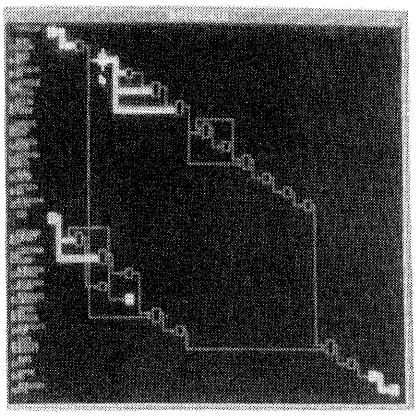

図7 データアグリゲーションとその可視化
Fig. 7 Data aggregation and the visualization.

アプリケーション情報の詳細で正確な処理順序を大局的に得ることである．1つ以上のサブプログラムに実現されている“開かれた”機能を見つけることも支援することを考える．

プログラム制御フロー図の場合，ユーザが，関心のあるプログラム要素を選びユニットデータ（式(19)）として与え，それに1つの色（ユニットカラー）を割り当てる．可視化システムの方では，式(20)から，素変数をすべて求め，式(21)に従ってすべての素要素を参照命令文ノードを求め，指定色（ユニットカラー）でマーキングする．さらに，プログラムラベルやベーシックブロックを制御フローのノードとするとき，2つのノード結ぶ有向エッジもユニットデータ参照を持つことが可能になる．

制御フローとプログラムリストは，2つのウィンドウに連動表示される．ユーザは，色付きノードの命令文をすぐに見つけることができ，これから，隠れた機能範囲や変数参照の因果関係が簡単に把握できる．

サブプログラム呼出関係図の場合，参照単位は，サブプログラムになる．同様に，サブプログラム間にわたる隠れた機能や変数参照の因果関係の把握ができる．

11. S/370 アセンブリープログラム分析/可視化事例

これまでに述べたアプローチに従って，“ソフトウェアマイクロスコープ”（SMS）プロトタイプを作成した．以下では，このツールを用いたアセンブリープログラムの事例を述べる．

SMSを利用して，図3のプログラムリストを処理すると，**図6**の基本表示画面を表示する．左側上部のウィンドウにプログラムテキストを，右側上部ウィンドウにプログラム全体制御フローを，右側下部ウィンドウにサブプログラム呼出関連図をそれぞれ表示する．

他に，テキスト表示として，$\mathcal{N}$, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{E}$, $\mathcal{D}$, $\mathcal{P}$ を表示することができる．ここまでは，ユーザは命令定義基本テーブルの設定作業を行うだけで，すべてSMSが自動処理を行う．

不完全で明示的な情報，または非明示的情報の可視化例として，レジスタ使用がある．図6の画面左側下方のウィンドウに，データユニットとして“R14”（レジスタ14）とサブプログラム名を，ユニットカラーとして“黄”と“赤”色をそれぞれ対応させて入力し，表示要求をする．その参照をフロー図上で見ることができ，**図7**にその結果を示す．

左側上部に現れた黄色いマークは，機械語命令コードにも書かれていないR14の使用があることを示し，右側下部に現れた黄色いマークは，リターンマクロ内部でR14を使用し，これら以外のマークは，サブプログラム呼び出しのリンクにR14を陽に使用していることを容易に確認できる．特に，最初の黄色いマークは，STM命令によるもので発見が難しい例である．

赤色ノードは，サブプログラム名定義・参照を表し，左中程左側にサブプログラムの開始点があることが分かる．他の赤いノードは，このサブプログラムへの呼び出し文があることを明示している．

約10^4行の大規模複雑プログラムの1例を示す．これは，長い間その全体構造がはっきりと分からなかったプログラムである．**図8**はその基本表示を示す．

ここまでは，ユーザは，命令定義基本テーブルに命令文定義作業を行うことだけである．

ソースコードと全体制御フローグラフ表示ウィンドウの連動を利用して，全体制御フローグラフの最上部のフロー線に沿って10～20回のノード選択をマウ

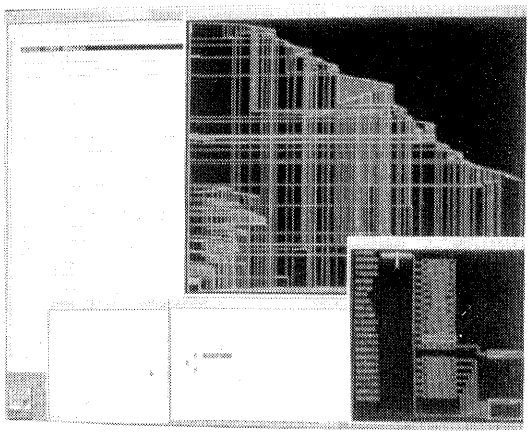

図8 レガシープログラムの可視化
Fig. 8 Basic visualization of a legacy program.

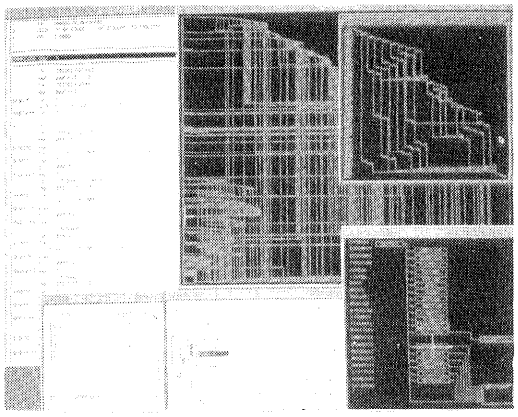

図9 サブプログラムの制御フローの可視化
Fig. 9 Visualization of subprogram control flows.

スクリック操作で行うと，正常処理機能フローがあることを簡単に知ることができた．それ以外の線は，ほとんどが例外処理対応であることも容易に分かった．

最左端に多くのノードが現れているが，ほとんどがデッドコードの開始点であり，マクロ命令文内のコードであることも分かった．また，サブプログラム機能間の干渉が多く存在することが一目で判断がつく．

新しい機能をこのプログラムに追加したいとき，その場所を，プログラムの初めから条件文の鎖を追いかけることにより，容易に見つけることができる．これまでの作業をプログラムエディタとメモをとる作業で行うならば，おそらく膨大な作業を行わなければならない．

一方，サブプログラムごとの制御フローを，ウィンドウに表示して見ることができる．画面右側下方に表示されているサブプログラム呼出図の中で1つのサブプログラム名を選択し，“Open” 要求を行う（実際に式 (17) を実行する）と，**図9**が得られる．画面右側上部に新たに開かれたウィンドウが，その制御フロー図を表示する．

以上に述べたツールの概略を参考にして，1,000 ステップのレガシーアセンブリープログラムの分析・理解に要する操作量が，通常エディタを利用する場合に比べて 10^{-3} のオーダーに減ることを述べる．この大きさより大きいレガシーアセンブリープログラムは通常理解困難と考えられる一方で，設計文書等プログラム以外の情報でその構造を知る方法はないことが多い．また，プログラムの実行可能命令全体の約 20% が条件ブランチ命令であることも経験的に知られている．この場合について，**基本分析**と**理解のための分析**に分けて比較を行う．ここで，サブルーチン等の必ずしも存在するとは限らない可変部分の分析は除く．これで，SMS ツール使用の場合のワークロードに対する通常エディタを使用する場合のワークロード比は，ここで求める比より大きくなることはない．

基本分析:データ処理命令文のタイプには，RR, RX, S, SS 命令等があるが，プログラム平均として RX タイプで代表すると，1命令あたり，操作コード，第1，2オペランド情報，基底&インデックスレジスタと変位情報等，全12種類の情報を記録する必要がある．これには通常エディタを用いると，12回の記録操作が行われる．一方，プログラム全体の制御フロー図を得るためには，1条件ブランチ命令あたり2つのブランチ先ラベルを記録するので，その操作が2回，フロー解析とノード配置を行う操作が（今回のアルゴリズムに従うとして）7回，描画が平均2本の出力線を書くとして最低3回の操作が要る．したがって，以上の合計の操作回数（量）は，$0.8\,\mathrm{K} \times 12 + 0.2\,\mathrm{K} \times 12 = 12\,\mathrm{K}$ となる．これの約3分の1がさらに検証作業として追加されると想定すると，総操作量は 17 K 回となる．

SMS ツールは，プログラムの使用するマクロがマクロ記述テーブルに定義されていれば自動処理を行う．未定義マクロを検出すると，分析初期段階でユーザにそのマクロ操作コード一覧を示してマクロ定義作業を促す．1つのマクロをマクロ記述テーブルに定義する操作量は，未定義マクロ情報を含む診断ファイルのオープン操作（1回），マクロ記述テーブルを開き入力可能域への位置合わせ（2回），マクロとそのパラメータの入力と属性指定（3回：マクロ操作コード入力，属性マーキング，パラメータ指定）であり，さらに，開いたウィンドウのクローズ（2回）等の操作があげられる．今，平均4マクロを定義する場合を考

えると，計17回の操作量となり，これは，上で概算した回数に対して 10^{-3} のオーダーに相当する．

プログラム理解：ここでは，データ変数参照関係とサブルーチン呼び出し順序の把握，の2ケースを検討する．従来方式では，全体フロー図の生成を行う必要があることから，操作量は，最低でも2.4Kの操作量に30%の検証・再作業量を加えて約3K回となる．さらに，色付けの操作量が追加になる．

ツールを使用する場合の操作は，データ変数名を1つの色番号に割り当てた後，“フロー図の色付けをする”操作アイコンをクリックすると，フロー上の参照ノードにその色が付く．この操作は3回であるので，従来方式との比は約 10^{-3} のオーダーとなる．次にサブルーチン呼び出し順序の把握の場合も，データ変数名の代わりにサブルーチン名を設定すればよいことが分かるので，操作量の比もやはり 10^{-3} 程度となる．他の場合もほぼ同様の比になる．

通常エディタを利用する場合，規模が小さくても比較的複雑なプログラムの理解のためのワークロードは，特に他人が書いたり保守したプログラムに関して増大し，以前に述べたようなサブルーチンが互いに入り繰り合う場合には，さらに増大する．たとえば，約30行の複雑なプログラムを分析理解するのに約2週間を要した例もある．これらの現象から，大規模で複雑度が増しているレガシーアセンブリープログラム理解は，SMS利用により 10^{-3} のオーダー以下に減ずることが可能になる．

12. 関連研究

アセンブリープログラムの分析可視化技術の研究は，他の言語で書かれたプログラムの場合が参考になる．初めに，それらについて述べる．

プログラム分析技術の研究は，プログラム理解を目的として，CとCobolプログラムについて盛んに行われている[12)~16)]．そのほかに，たとえば，デバッガ使用支援のためにプログラム分析利用を考える場合もある[16)]．これらの分析記述法は，ほとんどが，**ソースコード表現の特定情報**に着目し，プログラム構造，または処理（意味）情報を抽出する方法を述べている，また，ユーザが，対話的に必要なプログラム情報をツールに入力することが前提となっている[12),13),15)]．これらの研究においては，個々の目的とその実現方法について調べ，プログラム分析技術の可能性を議論している．前にも述べたように，プログラムに内在するプログラム要素間の関係情報の種類は大変多く，どれだけ多く正確に認識検出しプログラマーの利用に提供することが最終的には重要になる．

プログラムスライシングに基づく理解支援方法が提案されている．変数の設定・読み出し関係を命令文や制御フロー表現上で可視化することにより，プログラム検証やデバッグに有効であると考えられる[18)]．しかしながら，プログラム全体から見るとき，つねに一意的に決められる保証はない．プログラムへの入力データに依存する場合もある．新しい分析技術研究は，これらの個々の有効性を踏まえ，特に，複数プログラミング言語を利用するシステムプログラム分析を考えるためには，一般性のある技術を目指す必要がある[13)]．

保守・再利用を目的としたプログラム構造情報抽出と表現技術の研究は，大別して，グラフとテキスト形式で表現する2通りがある[1),12),14),15)]．テキスト形式表示アプローチでは，プログラムの意味的仕様の記述が目的なので，新しい記述方法の提案とその記述言語を利用する方法を導入する必要がある．

グラフ表現は，いろいろな表現方法が提案されている[3),5),17)]．これらのグラフノード配置方式は，Sugiyama-Tagawa-Todaのアルゴリズムが基本になっていて[17)]，“直感的見やすさ”を重視している．たとえば，“ノードを結ぶ結線の交差量をできる限り少なくする”等のノード配置ルールを設け，全体として，それらのルールを満たすようにノード再配置を繰り返す．単純グラフであれば見やすい表示を生成する．

しかしながら，この一般的な配置方式アプローチは，ノード配列空間をまったく規程しないので，大規模複雑プログラムの制御フロー図等を，必ずしも“直感的に見やすく”表示しない．1つのノードが表示画面中央に配置されているとして，そのノードと他のノードとの関係について判断基準となる目安がないからである．

アセンブリープログラム理解支援技術について，以上に述べたアプローチに類似した研究[2)]と，プログラムの複雑度を命令タイプとオペランド数等の情報をパラメータとして，計算によって直接示す研究がある[6)]．特徴は，コードレベル分析が中心で，制御フローの可視化をそれほど実現していない．レジスタの使用分析は，他言語プログラムにはあまり見られないが，基本的には，共有，またはグローバル変数の取扱いをすればよいので，本質的には変わらない．問題は，レジスタの内容が分からないと分析不可能になる場合があり，何らかの解決方法を見つけなければならない．一般に，プログラム理解に必要なプログラム情報を求める計算は，アセンブリープログラムの方が多い．

現在までのところ，すべての大規模レガシープログラムに適用できるツールがなく，ツール適用不可能な

プログラムが存在する場合，そのプログラムを含むソフトウェア全体の分析は大変難しくなる．これは，ソフトウェア保守にCASEツールを導入するうえでの決定的疎外要因であり，解決が求められている．

13. まとめ

この論文では，プログラム理解プロセスモデル，プログラム情報モデル，プログラム情報の解析的記述方法を述べて，それらに基づいて，プログラム情報モデルの可視化を述べた．

プログラムリストは，アプリケーション情報を非明示的情報と不完全明示的情報の形態で含んでいる．これは保守作業の品質に決定的影響を与えるので，これらの情報を完全で明示的な情報に変換し，プログラム構造を，時間（実行順序）軸と選択軸の2次元空間で可視化し，理解を容易にする必要がある．

一方，プログラム分析アルゴリズムは，通常ソースコード表現を用いて記述されることが多いが，プログラム分析技術とするためには，プログラム要素間に埋められている多くの情報を正確に一般性を与えて表現するアプローチが必要である．ブラケット記号を用いる記述法は，プログラム情報を正確に記述するとともに記述された表現を分析変換できる能力を持つ．

膨大な手作業・メンタルワークを排除し，大規模複雑プログラム理解が可能になったとき，レガシープログラムの保守問題はなくなる．プログラム情報の解析的記述方法とソフトウェアマイクロスコープは，これを解決できる可能性を示した．

謝辞 本研究にあたって，日本アイ・ビー・エム株式会社伊藤隆研究員とプロトタイプを共同開発し，初期の研究においては，同社藤井邦和，牧野正士，四野見英明，川副博主任研究員が共同しました．また，査読者から，本論文を良くするための丁寧なコメントをいただきました．ここに感謝いたします．

参考文献

1) Biggerstaff, T.J., Hopkins, T.J. and Webster, D.E.: DESIRE: A System for Design Recovery, MCC Tech. Mem. STP-081-89, Apr. (1989).
2) Chen, S., Heisler, K.G., Tsai, W.T., Chen, X. and Leung, E.: A Model of Assembly Program Maintenance, *Journal of Software Maintenance: Research and Practice*, Vol.2, No.1, pp.3–32 (1990).
3) Kamada, T. and Kawai, S.: A General Framework for Visualizing Abstract Objects and Relations, *ACM Trans. Graphics*, Vol.10, No.1, pp.1–39 (1991).
4) Gansner, E.R., Koutsofios, E., North, S.C. and Vo, K.-P.: A Technique for Drawing Directed Graphs, *IEEE Trans. Softw. Eng.*, Vol.19, No.3, pp.214–229 (1993).
5) Baker, M.J. and Eick, S.G.: Visualizing Software Systems, *Proc. 16th International Conference on Software Engineering*, pp.59–67 (1994).
6) Boydston, R.: The Effect of Program Complexity on Programming Productivity and Program Quality, IBM Santa Teresa Laboratory, TR 03.071 (1979).
7) 秋山義博：大規模複雑プログラム理解のためのプログラム分析と可視化技術，情報処理学会研究報告，ソフトウェア工学 102-6, Jan. (1995).
8) 秋山義博：ソフトウェアマイクロスコープ，情報処理学会研究報告，ソフトウェア工学 106-7, Nov. (1995).
9) Enterprise System Architecture/390, Principles of Operations, SA22-7201-00, October 1990, IBM Corporation.
10) IBM High Level Assembler/MVS & VM & VSE Language References, SC26-4940.
11) IBM High Level Assembler/MVS & VM & VSE Programmer's Guide, SC26-4941.
12) Ning, J.Q., Engberts, A. and Kozaczynski. W. (Voytek): Automated Support for Legacy Code Understanding, *Comm. ACM*, Vol.37, No.5, pp.50–57 (1994).
13) Markosian, L., Newcomb, P., Brand, R., Burson, S. and Kitzmiller, T.: Using and Enabling Technology to Reengineer Legacy Systems, *Comm. ACM*, Vol.37, No.5, pp.58–70 (1994).
14) Biggerstaff, T.J., Mitbander, B.G. and Webster, D.E.: Program Understanding and the Concept Assignment Program, *Comm. ACM*, Vol.37, No.5, pp.72–82 (1994).
15) 原田 実，吉川彰一，永井英一郎：Cobolプログラムから非手続き仕様を逆生成するリバースエンジニア：Core/M，情報処理学会論文誌，Vol.36, No.3, pp.714–728 (1995).
16) 佐藤慎一，飯田 元，井上克郎：プログラムの依存関係解析に基づくデバッグ支援ツールの試作，情報処理学会論文誌，Vol.37, No.4, pp.536–545 (1995).
17) Eades, P. and Sugiyama, K.: How to Draw a Direct Graph, *Journal of Information Processing*, Vol.13, No.4, pp.424–437 (1990).
18) Gallagher, K.B. and Lyle, J.R.: Using Program Slicing in Software Maintenance, *IEEE Trans. Softw. Eng.*, Vol.17, No.8, pp.751–761 (1991).
19) Tarjan, R.: Finding Dominators in Directed

Graphs, *SIAM J.Comput.*, Vol.3, No.1, pp.62–89 (1974).
20) Sugiyama K., Tagawa S. and Toda M.: Methods for Visual Understanding of Hierarchical System Structures, *IEEE Trans. Sys. Man & Cyb.*, Vol.SMC-11, No.2, pp.109–215 (1981).

(平成8年7月17日受付)
(平成9年3月 7 日採録)

秋山 義博（正会員）

1969年東北大学理学部物理学科卒業，1971年金沢大学大学院修士課程修了，1974年東京都立大学大学院博士課程修了．理学博士．同年4月日本アイ・ビー・エム（株）入社．東京基礎研究所，システム研究所，および米国IBMサンタテレサ研究所等において大規模システムソフトウェア開発とソフトウェア工学技術の研究開発に従事．1994年より金沢工業大学教授．現在，オブジェクト指向OS，マルチメディアやファジー計算，モーバイル情報システム技術，ソフトウェア工学の研究に関心を持つ．「仮想計算機」（共著，共立出版）．IEEE，ACM各会員．

水野 舜（正会員）

1966年立教大学理学部物理学科卒業．1972年京都大学大学院博士課程修了．同年金沢工業大学講師．助教授を経て，1984年同大教授．理学博士．コンピュータを用いた障害者補助に興味を持つ．画像処理や自然言語処理の研究に従事．電子情報通信学会，日本ファジィ学会，テレビジョン学会，IEEE各会員．