25M－08

「車載ソフトウェアの品質向上のための ソースコードの見える化の実現法」

猪股俊光（ソフトウェア情報学部　教授）、
福原和哉（ i-MOSプロジェクト研究員）、
高橋耶真人（ソフトウェア情報学研究科　博士前期課程、現在　株式会社リコー）

車載ソフトウェア開発では、既存のソースコードを改変して、他の車種用にカスタマイズすることが行われる。そのときに、元のソースコードの一部の変更が他のコードのどの部分に影響を与えるのかが明らかになれば、コードの改変によるバグ（誤動作の原因）の発生を未然に防ぐことができ、品質向上が期待される。そこで、本課題では、ソースコードで表された処理の流れやデータの更新過程を視覚化（見える化）することで、コード改変の影響範囲を明らかにすること目指し、そのための表現法を考案するとともに、それに基づきソースコードを可視化するツールを試作した。

1　研究の概要

車載ソフトウェアに代表される組込みソフトウェア製品ではモデルチェンジのほか上位や下位のモデルの開発が頻繁に行われるため、差分開発や派生開発が主流となっており、新規開発を行うことは少ない。差分開発や派生開発では既存のある時点での製品ソースコードを元に機能拡張や統廃合を行うことで開発が行われる。 そのため、コードレビュー[1,2]などを通じて、コードの変更作業が製品の動作に与える影響を正しく理解し、十分に注意を払いながら行う必要がある[3]。しかし、実際には派生元となるソースコードの作成者とは別の開発者が開発を行うことが多く, 製品ソースコードの構造などを把握していない技術者が変更を行っている。また、ソースコードに変更を与えた場合の影響範囲の特定には、ソースコードに含まれている変数や関数の相互関係が必要となるため、目視による検査で影響範囲を明らかにすることは困難である。コードレビューで用いられるソースコード可視化ツールとして代表的なGNU Global, Ctags, Doxygen では構文の字面から読み取ることが可能な変数と関数間の関係については図示等をしてくれるものの、値等の伝搬によって定まる関係については図示されない。

そこで、本研究では、プログラムに含まれている関数や変数の相互関係を視覚化することが、プログラムに対する追加修正が与える影響範囲を特定することなどに役立つものと考え、関数や変数の関係の各表現方法について検討した。その結果、関数どうしの関係、関数と変数の関係、変数どうしの関係についてプログラム開発者やレビュー担当者の理解を助けるために可視化する実現法を提案[4]するとともに、それに基づきソース コード内の要素間の関係をグラフとして可視化するツールを試作し、評価を行った[5]。その結果、ある条件を満たすソースコードに対して、関数・変数関係の可視化が可能であることが確認された。

2　研究の内容

2.1　対象とするプログラム

対象とするプログラムは、C言語で書かれたソースコードで文法エラーが含まれていないものとする。ただし、本研究では次の構文は対象外とする。

・配列、構造体、共有体といった構造
・関数ポインタ
・代入文における多重間接参照
　（例えば,x=**p;）

なお、これらの制限の取り扱いについては3.2節で述べる。

2.2 プログラムグラフ

対象とするプログラム P の中に含まれる関数と変数の集合をそれぞれ F, V とする.

$P = (F, V)$

$F = f1, f2, \cdots, fn$

$V = V_{f1} \cup V_{f2} \cup \cdots \cup V_{fn} \cup V_g$

Vfi ：関数 fi の仮引数および、局所変数の定義

Vg：大域変数の全体

$P = (F, V)$ のもとで、関数や変数に関する三項関係 $R_{FV}, R_{FF}, R_{VV}, R$ をそれぞれ次のように定める。

関数と変数の関係：$R_{FV} \subseteq F \times V \times O_{FV}$

O_{FV} = {R, W, RefR, RefW}

関数どうしの関係：$R_{FF} \subseteq F \times F \times O_{FF}$

O_{FF} = {call, callR, callW}

変数どうしの関係：$R_{VV} \subseteq V \times V \times O_{VV}$

O_{VV} = {Ref, W}

$R = R_{FV} \cup R_{FF} \cup R_{VV}$

以上のことからプログラムグラフ G を次式とする。

$$G = (P, R)$$

2.3 関数と変数の関係 R_{FV}

対象とするソースコード中の関数 f とその中の代入文に現れる変数 v の間に次の関係 R_{FV} が成り立つことを $f \rightarrow O_{FV}\ v$ と書く。

R 変数 v が代入文の右辺や仮引数、実引数、条件式に出現：$f \rightarrow$**R** v

W 変数 v が代入文の左辺に出現：$f \rightarrow$**W** v

refR 変数 v が間接参照演算子をともない $*v$ として代入文の右辺や仮引数、実引数、条件式に出現：$f \rightarrow$**refR** v

refW 変数 v が間接参照演算子をともない $*v$ として代入文の左辺に出現：$f \rightarrow$**refW** v

2.4 関数どうしの関係 R_{FF}

ソースコード中の関数 f と関数 f' の関係 R_{FF} を

$f \Rightarrow O_{FF}\ f'$ と書く。

Call f の中で、関数 f' を値呼びしている：$f \Rightarrow$**Call** f'

CallR f の中で、関数 f' を参照呼びしている（実引数は更新されない）：$f \Rightarrow$**CallR** f'

CallW f の中で, 関数 f' を参照呼びしている（実引数は更新される）：$f \Rightarrow$**CallW** f'

2.5 変数どうしの関係 R_{VV}

ある関数の中の代入文、実引数、仮引数に現れる変数の間に成り立つ関係 R_{VV} では、関数の実引数は代入文の右辺に現れる変数 r 、仮引数は代入文の左辺に現れる変数 l とそれぞれみなし、関数内の変数を l と r に分類する。

Ref ポインタ変数と、それが指し示す変数との関係

- rに間接演算子が用いられておらず l の評価値がアド レス値のとき、rが指し示す各変数 r_i について $\boldsymbol{l} \rightarrow$Ref $\boldsymbol{r}_i$と書く。
- rにアドレス演算子が存在するとき、または、lに間接演算子が用いられ、左辺 lの評価値がアドレス値のとき、 $\boldsymbol{l} \rightarrow$Ref $\boldsymbol{r}$と書く。
- rに間接演算子が用いられているとき、r が指し示す変数が指し示している各変数 rついて $\boldsymbol{r} \rightarrow$Ref $\boldsymbol{rr}_i$と書く。

W 左辺に現れる変数の関係

- 代入式内に間接演算子、アドレス演算子が用いられておらず、 l の評価値がアドレス値でないとき $\boldsymbol{r} \rightarrow$**W** $\boldsymbol{l}$ と書く。
- l が間接演算子を用いており、 l の評価値がアドレス値ではないとき、l が指し示す各変数 l_k ついて、$\boldsymbol{l} \rightarrow \boldsymbol{W}\ \boldsymbol{l}_k$と書く。

【例１】 ソースコードの例

```
1 : int g, *gp;
2 : void foo(int a1) {
3 :   baz(&a1);
4 :   gp = &g;
5 :   *gp = bar(&a1, *gp);
6 : }
7 : int bar(int *b1, int b2) {
8 :   *b1=b2*2;
9 :   if(b2!=0){
10 :     return bar(b1, b2 - 1);
11 :   } else {
12 :     return *b1;
13 :   }
```

```
14 : }
15 : void baz(int *c1) {
16 :     g = *c1;
17 : }
18 : void main() {
19 :     foo(2);
20 : }
```

例１のソースコードに対するプログラムグラフを図１に示す。図中、関数*f*は四角、変数*v*は楕円でそれぞれ表している。また、関係の種類（O_{FV}, O_{FF}, O_{VV}）は矢印のラベルとし、とくに、R_{FF}については矢印を白抜きで表している。

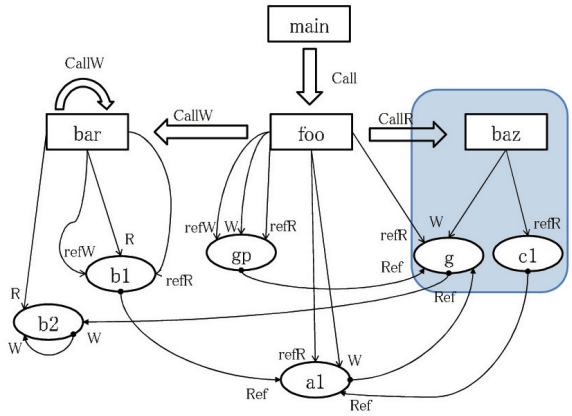

図１　例１のプログラムグラフ

2.6　可読性を考慮した関係の表現

（１）　表記法

関数や変数の関係 R_{FV}, R_{FF}, R_{VV} から得られたプログラムグラフ *G* と、ソースコードとは必ずしも１対１に対応しておらず、コードレビューにとっては可読性が悪い。

そこでプログラムグラフ *G* として表された関係と構文情報をもとに,可読性を考慮した表現形式を考案した。 C 言語では、変数や関数の利用は式でのみ許されていることから、変数や関数の関係は式単位で表すことにし、関数もしくは変数に着目した場合の関係の表現形式を、

［式中での使われ方 => 式の使われ方］

という形で表記する。

（２）式中での使われ方

式中で着目した構文要素に対する演算の種類を以下に示す表記で表すものとする。一般的に、式中には複数の演算が含まれるため、表記は演算の実行順に従って左から並べる。

演　　式中での二項演算もしくは単項演算

***(*n*)**　式中での間接参照

n は間参照後のポインタの多重度を示す。たとえば int **a; と定義されている a に対して１段階の間接参照を行う式 *a から得られる表記は *(1) となる。

&(*n*)　式中でのアドレス演算

n はアドレス演算後のポインタの多重度を示す。たとえば int *a; と定義されている a に対して１段階のアドレス演算を行う式 &a から得られる表記は &(2) となる。

前　式中での前置式の演算

後　式中での後置式の演算

代　式中で代入式の右辺式として利用

例えば、次の代入文における変数 a の使われ方は次のようにして表される。

【例２】　　y = *a++ + 3;

例２の代入文を一時変数 T1～T4を用いて次のように評価順序に従って分解し、表記との対応付ける。

```
T1=a++     …後
T2=*T1     … *(0)
T3=T2+3    …演
T4=y=T3    …代
```

これより、変数 a の代入文中での使われ方は、

（後, *(0), 演, 代）

と表現される。

（２）式の使われ方

“式中での使われ方”だけを可視化したのでは、コードが示す意図の見落しにつながり、式の演算結果の利用目的はコードの理解に非常に有益な情報である。そこで、“式の使われ方”について次のように区別して表示する。

Write　間接参照を伴わない代入文の左辺として用いられ、何らかの値が直接書き込まれる。

RefWrite　間接参照を伴う代入文の左辺として用いられ、何らかの値が間接参照先に書き込まれる。

破棄　副作用のみを目的とし、式の値は何にも

用いられない。関数呼び出しの場合には関数の戻り値が存在しない、もしくは使われない。

初期化　式の値は変数の初期値として用いられる。

オフセット　間接参照演算で用いるオフセット値として用 いられる。

Condition　条件文や制御文の条件式部分、もしくは、Switch 文の評価式部分で用いられる。

実引数　関数呼び出しの実引数として用いられる。

Return　Return 文の引数として用いられる。

これらを用いた関係表記の導出例を次に示す。

【例３】　return (*a++ + 3);

a を含む式の値は return 文の引数に用いられていることより右辺の関係は Return。また、変数 a の使われ方より左辺は（後, *(0), 演）。よって、変数 a の関係は、次のとおり。

［（後, *(0), 演）=> Return］

【例４】　*(a+4) = b;

変数 a を含む式が代入文の左辺に出現しており、間接参照を伴っていることから右辺は RefWrite。a の使われ方は（演, *(0)）。よって、a の関係表記は次のとおり。

［（演, *(0)）=> RefWrite］

2.7　関係可視化ツール

（１）概要

提案した関係の表現を用いてソースコード中に含まれる要素間の関係を図示するツールを試作した。その処理フローを図２に示す。

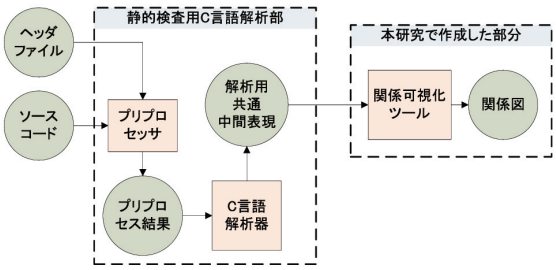

図２　関係可視化ツールの処理の流れ

試作したツールは C 言語のソースコードを入力とし、関係にしたがった図表現を HTML形式のテキストファイルとして出る。HTML からグラフ画像の生成には Graphviz を、C 言語のソースコードの解析には筆者らが開発している、コードレビュー支援を目的とした静的検査ツール[6]のプリプロセッサと解析器を用いた。

試作したツールでは、注目したい変数や関数に着目した図示が行える。利用者は、図３のようにブラウザ上に表示された関数名や変数名の中から着目したい項目をクリックすることで、その項目に関する関係図が閲覧できる。

例５のソースコードに対する出力結果を図４と図５に示す。

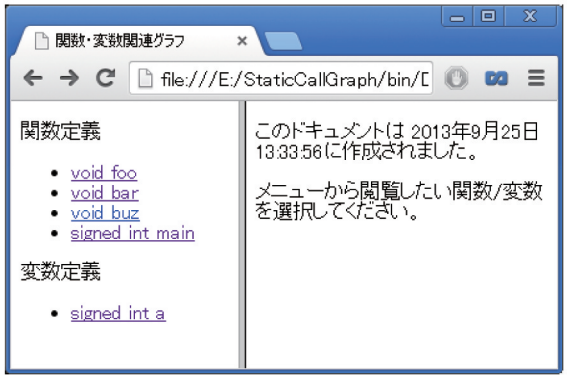

図３　関係可視化ツールの画面例

【例５】　ソースコードの例２

```
1 :  int a=0;
2 :  void foo(int *p) {
3 :        *p = 1;
4 :  }
5 :  void bar(int *p) {
6 :        foo(p);
7 :  }
8 :  void baz(int *p) {
9 :        bar(p);
10 :  }
11 :  int main() {
12 :      baz(&a);
13 :      return 0;
14 :  }
```

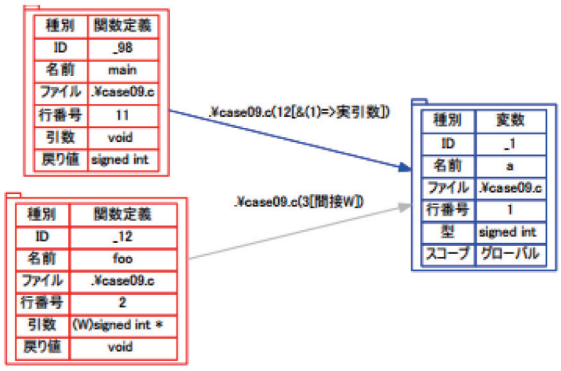

図４　変数 a の関係図

図５　関数 foo の関係図

（２）変数の関係図の概要

変数の関係図では注目する変数が右側に、アクセスを行う関数や関係をもつ変数が左側に図示される。例えば、図４の変数関係図からは変数 a について、次のような関係を読み取ることができる。

(12[&(1)=>実引数])

main関数の12行目で変数 a の参照が実引数へ渡される。

(3[間接 W])

関数 foo によって変数 a は３行目で間接的に値が変更される。このようにプログラムグラフをもとに、特定の変数や関数に着目した関係を解析することで、引数を経由 した間接的なアクセスを可視化することができる。

（３）関数の関係図の概要

関数の関係図では注目したい関数を中心として左側に関数を呼び出している関数を、右側には関数が読み書きを行う変数と呼び出しを行う関数が図示される。

例えば、図５の関数関係図からは次のような関係を読み取ることができる。

・関数 foo は６行目で関数 bar に呼び出される。
・関数 foo は３行目で間接的に変数 a に値を書き込む。

3　これまで得られた研究の成果

3.1　実行例

試作したツールを用いることで、図３～図５に示したように、入力されたソースコードからプログラムグラフを生成することが可能となった。

ある程度の規模のソースコードとして、16/32bit MCU 向け RTOS のソースコード（1666行）を入力としたときのプログラムグラフの生成にかかる時間（CPU 2.5GHz）を次表に示す。

ファイル数	11
ステップ数	1666行
実行時間	3.16秒

実際の組み込みソフトウェアなどでは、より多くの関数や変数が出現することが多いため、生成される関係図は巨大になる。これに対しては、必要としない関係を隠す機能の実装や、関係情報を表形式で出力することが有効である。

3.2　考察

（１）プログラムグラフによる変更波及解析

プログラム P のあるコードが変更（追加、修正、削除）されたことが、プログラム中のどこのコードにどのような影響を与えるかを分析すること、すなわち、変更波及解析を、本課題で導入したプログラムグラフ $G = (P, R)$ を用いて行うことができる。例えば、ある関数を変更したときには R_{VV} を関係 R_{FV} や R_{FF} をもとに、ある変数を変更したときには R_{FV} を関係 R_{VV} をもとに影響範囲を調べることができる。

（２）複合型への対応

現時点では対象としていない配列などの複合型の内部要素の関係を出力する場合、それぞれを個別の変数として出力すると、複合型という単一の型としてみなす場合に比べて、解析する際の精度が向上する。その場合、同一の複合型に含まれる要素の関係を改めて図示する必要がある。

（3）多重間接参照

二項演算を含まない代入文であれば、一時変数を用いることにより、多重間接参照を用いない代入文に置き換えることが可能である。例えば、v = **p; は、

tmp = *p; v = *tmp

と置き換えられる。この場合、オリジナルのプログラムには含まれていない変数 tmp が導入されることになる。

（4）たらいまわしにされる参照

例５のコード例のように、ある関数に与えられた引数を別の関数の引数として渡すことがしばしば行われるが、試作ツールでは最終的に利用される時点のみが 図示される。しかし、引数として渡される過程が知りたい場合もある。これについては各関数において引数を通じて伝搬する値の解析が必要である。

4　今後の具体的な展開

今後、取り組むべき課題は次のとおり。

- 変更波及解析への応用
- 複合型（配列、構造体）への対応
- たらいまわしにされる参照への対応
- 多重間接参照への対応

5　論文・学会発表等の実績

- 高橋耶真人、福原和哉、猪股俊光、新井義和、今井信太郎、“プログラムの関数・変数関係の一表現法”、信学技法, Vol. 113, No. 269, SS2013-40, pp. 49-54, 2013.
- 福原和哉、高橋耶真人、猪股俊光、新井義和、今井信太郎、“コードレビュー支援システムのための関数・変数関係の可視化実現法”、信学技法, Vol. 113, No. 269, SS2013-41, pp. 55～60, 2013.

6　受賞・特許

なし

7　その他

[1]SQuBOK 策定部会：ソフトウェア品質知識体系ガイド、 SQuBOK Guide、オーム社、2007

[2]Cem Kaner, Jack Falk, Hung Quoc Nguye：基礎から学ぶソフトウェアテスト、日経 BP 社、2001

[3]独立行政法人情報処理推進機構技術本部ソフトウェア・エンジニアリング・センター：SECBOOKS 組込みソフトウェア開発における品質向上の勧め［テスト編 事例集］（SEC books）、独立行政法人情報処理推進機構、2012

[4]高橋 耶真人、福原 和哉、猪股 俊光、新井 義和、今井 信太郎：“プログラムの関数・変数関係の一表現法”、信学技報 SS、2013

[5]福原和哉、高橋耶真人、猪股俊光、新井義和、今井信太郎：“コードレビュー支援システムのための関数・変数関係の可視化実現法”、信 学 技 法、Vol.113, No. 269, SS2013-41, pp. 55～60, 2013.

[6]福原 和哉、高橋 耶真人、猪股 俊光、新井 義和、今井 信太郎：“組込みソフトウェア向けコーディング規約チェッカのためのカスタマイズの一方式”、FIT2012第11回情報科学技術フォーラム, 2012