6T−1

意味構成管理システムを用いた修正プログラムにおける相互干渉の検出

直井 邦彰　高橋 直久
NTT ソフトウェア研究所

1 はじめに

プログラムを複数の編集者が独立して修正した後、各修正結果をマージした版を生成すると、複数修正間での相互干渉のためにプログラムが意図しない動作をすることがある[1]。我々は、Moriconi らの提案した情報フローモデル[2]を基礎に、意味的誤りの検出と解決を行なう意味構成管理システムの構築を進めている。Horwitz らの提案した従来の相互干渉検出法[3]では、検出した影響波及領域に実際には影響が及ばない変数を含むことがあり、それらの除去が難しいために相互干渉の検出精度向上が困難である。ここでは本問題の解決を目指し、以下の特徴を持つ高精度の相互干渉の検出モデルを提案する。

(1) 計算経路の解析を精密化することによって、実際に実行される可能性のある計算経路を絞り込む。
(2) 絞り込んだ計算経路に対する変数間の依存関係を求めることによって、影響波及領域を絞り込む。

2 意味構成管理システム

修正後のプログラムにおいて、修正前とは異なる文の系列で値が定められる変数を、修正による影響が及ぶ変数であるという。同一プログラムに対して独立に修正を施してできる複数の版をマージし、同一の版として保持すると(図2参照)、一方の修正が影響を及ぼす変数に他方の修正も影響を及ぼすのでプログラムが意図した通り動作しなくなることがある。このように、各修正に対する影響波及領域間で共通の要素が存在する時、相互干渉が発生したと呼ぶ[1]。従来の構成管理[4]は、利用者の意図した版の組合せを与えることを目的としているため、上記の相互干渉による意味的な誤りの検出能力は不十分である。Moriconi らは、プログラムの意味的な情報を用いて影響波及領域を検出する機構を備える構成管理を提案している[2]。我々はこれを基礎に、相互干渉の検出を可能とするとともに、影響波及領域の検出の精度を向上させた意味構成管理システムの構築を進めている。

3 従来の相互干渉検出法における問題

次に示すような相互干渉の検出法が、Horwitz らにより提案されている[3]。

(1) プログラムより、データの依存関係と条件文の制御構造を表現したシステム依存グラフ[5](SDG)を作成する。
(2) SDG から修正部分に依存する変数の集合を求める。
(3) 各修正について、(2)で求めた集合間で共通する要素を求める。

図1に示す例で、calc の ST15 を修正した場合、図3の SDG 中の破線で囲んだ領域に影響が及ぶ。その領域には ST19 が含まれるために相互干渉が発生したと判定される。しかし実際には4章で述べる通り、ST15 の変数 m から ST19 の変数 g への依存関係は存在しないため、相互干渉は発生していない。このように、相互干渉が発生していないのに発生可能性があると判定してしまうことがある。さらに次に示す理由により、計算経路の解析を詳細化して相互干渉の検出精度を向上させるのは困難である。

(1) SDG では計算経路の情報が陽には表現されていない。
(2) そのため、実行不可能な計算経路をたどる時にのみ存在する変数間の依存関係を検出することが難しい。

```
   Program Main
ST1    call Console_in(v, w, x, y, z);
ST2    call calc(v, w, x, y, z, s, t);
ST3    call Console_out(s, t)
       end
   Procedure calc(p, q, a, b, c, g, h)
ST11   g := 0;                  ST16   k := p - 1;
ST12   h := 0;                  ST17   n := m + m;
ST13   if p > 0                 ST18   if (k > 0 and q > 0)
         then                             then
ST14       m := b - a           ST19       g := n - 5
       else                              else
ST15       m := c + a           ST20       h := n + 5
       fi                                fi
                                         return
```

図1 サンプルプログラム

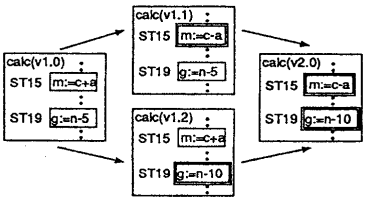

図2 図1の手続きcalcの修正に対する版間の関係

4 相互干渉検出モデル

上記問題の解決のため4.1,4.2で示す特徴を持つ相互干渉検出モデルを提案し、相互干渉の検出精度を向上させる。

4.1 計算経路の絞り込み

構文情報に従い実行順序をたどることにより得られる文の系列を、プログラムの計算経路という。これまでに、プログラムの記号実行により、ある計算経路が実行される可能性がある(FP:Feasible Path という)か否(IP:Infeasible Path という)かを調べて効果的なテストデータを作成する手法が提案されている[6]。我々はブレスブルガー算術[7]を用い、以下の(1)〜(3)の手順で IP を検出して FP を絞り込むことにより、影響波及解析の精度向上を図る。

(1) 計算経路に条件文が現われるたびに、各分岐に対応した文の系列をそれぞれたどることにより計算経路を求める。

例えば図1の calc から、図4の経路 P1 〜 P4 を求める。

(2) (1)で求めた各計算経路について、次のように IP を検出する。開始文からある条件文 A までたどる計算経路においてA とは異なる条件文が存在する時、それら条件文より

*Detecting an interference in a modified program with a Semantic Configuration Manager
Kuniaki NAOI, Naohisa TAKAHASHI
NTT Software Laboratories

得られる論理式および、論理式の計算に必要な経路上の文の集合を求める。さらに、この集合にプレスブルガー算術を適用する。ここでプレスブルガー算術とは、整数、整数上の変数、加減算、大小比較、ブールの論理演算子(and,or,not)および限定作用素(∀,∃)から構成される算術であり、この算術に含まれる論理式のうち自由変数を含まない論理式(プレスブルガー文)の真偽は一般に定まる。上で求めた集合に対して変数や関数呼び出しの置き換えにより、Aの条件判定の式の真偽を定めるのに必要なプレスブルガー文を求める。この文が真であると判定された場合に、Aの分岐によりできる計算経路からIPを取り除くことが可能となる。

例えば、図1の手続きcalcに対して、図3のSDGより条件文に関係する依存関係を図5に示す通り求めることができる。これより、ST11からST15を経由してST18までたどった計算経路は、ST18からthen方向へは実行されないことが以下の通り判定でき、P3がIPであると判定される。

論理式(p≤0)と文k=p−1の両方の条件を満たす時、論理式(k>0 and q>0)は必ず偽となると判定される。

(3) (1)で求めた計算経路から(2)でIPと判定された計算経路を取り除く。

4.2 影響波及領域の絞り込み

4.1で示した通りFPを絞り込んだ後、以下のように影響波及領域を求める。

(1) FPに対する変数間依存関係を求める。

(2) 情報フローモデル[3]を基礎に、求めた依存関係とFPとを用いて修正部分に依存する変数の集合を求める。

図1のST19がST15に依存するか調べる例を考える。ST15の変数mからST17の変数nへのフローをm(ST15)⇒n(ST17)と表した時、このフローとn(ST17)⇒g(ST19)とはFPの一部として存在するが、ST15からST19に至るFPは存在しない。このためm(ST15)⇒g(ST19)はFPには存在せず、ST19のgはST15のmに依存しないことがわかる。

4.3 モデルの特徴

従来の情報フローに基づく影響波及領域検出法では、SDGを用いた手法に比べ能力的に同等であるのに、手間が増えるという問題がある。一方SDGを用いた場合には、3章で述べた理由からIPの検出や、特定の計算経路に対する影響波及解析を行ないにくく、影響波及領域の検出精度向上が困難である。本稿で提案した手法では、情報フローモデルが計算経路の解析に適しているという点に着目し、IPを検出するとともにFPに対する影響波及解析を行なうことにより、影響波及領域の検出精度を向上させている。さらに相互干渉を複数の情報フローの衝突として捉えることにより、その検出機構を与えている。

5 おわりに

情報フローモデルを基礎に、計算経路の解析により影響波及領域を絞り込む機能を備えた高精度の相互干渉検出モデルを提案した。今後、このモデルに基づき、プログラムの意味構成管理システムの具体化を進展させる予定である。

最後に、日頃ご指導ご討論いただく、伊藤正樹リーダはじめグループの皆様に感謝いたします。

[参考文献]

[1] 直井, 高橋: 細粒度オブジェクト空間を用いたCASEデータベースのトランザクション処理モデル, 情処全大44,7H-3(1992.3).

[2] Moriconi,M. and Winkler,T.C.:Approximate Reasoning About the Semantic Effects of Program Changes, *IEEE Transactions on Software Engineering*, Vol.16, No.9, pp.980-992(1990.9).

[3] Horwitz,S., Prins,J. and Reps,T.: Integrating Noninterfering Versions of Programs, *ACM Transactions on Programming Languages and Systems*, Vol.11, No.3, pp.345-387(1989.7).

[4] Katz, R.H.:Toward a Unified Framework for Version Modeling in Engineering Databases, *ACM Computing Surveys*, Vol.22, No.4, pp375-408(1990.12).

[5] Horwitz,S., Reps,T. and Binkley,D.:Interprocedural Slicing Using Dependence Graphs, *ACM Transactions on Programming Languages and Systems*, Vol.12, No.1, pp.26-60(1990.1).

[6] Howden, W.E.:A Survey of Static Analysis Methods, *Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE*, pp.101-115(1981).

[7] 東野, 関, 谷口: 代数的仕様から関数型プログラムの導出とその実行, 情報処理, Vol.29, No.8, pp.881-896(1988.8).

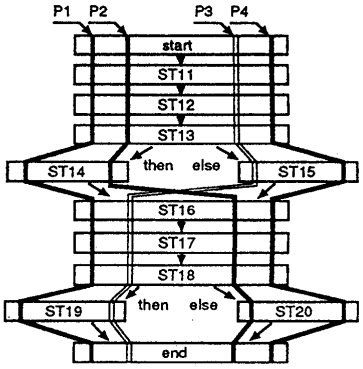

図4 手続きcalcに対する計算経路

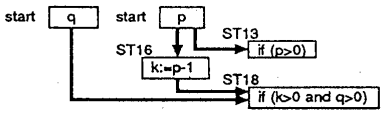

図5 計算経路の絞り込みで用いられる文とその関係

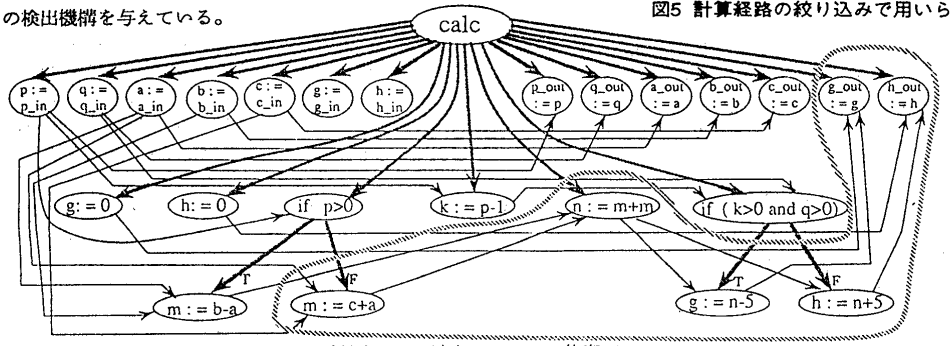

図3 手続きcalcに対するシステム依存グラフ