プログラムバグ潜在域　Critical Sliceの最適化技法

1J－2

下村　隆夫
NTTソフトウェア研究所

1．はじめに

手続き型言語を対象とし，プログラムバグ潜在域Critical Sliceを用いて，文の記述漏れを含む全ての種類のバグを文単位で検出できるバグ究明方式が提案されている[1]．プログラムを実行した時，ある変数値エラーが発生している場合に，Critical Sliceは値誤りバグ（式の誤りに関するバグ）の存在範囲を示すプログラム内の命令の集合である．本論文では，このCritical Sliceのサイズを最適化する技法について提案する．

2．Critical Slice

プログラムに含まれるバグを**値誤りバグ**と**設定漏れバグ**に分類する（表1）．ある入力を与えてプログラムを実行した場合，実行されたパス（命令の列）を実行系列と呼ぶ．t番目に命令の実行が行われた時点を実行時点tと呼ぶ．実行時点tで実行された命令で使用された（値が参照された）変数の集合をUse(t)で表わす．変数vに関して，2つの実行時点s, t（s < t）の間には，3つの依存関係，s ∈ Def(t, v), s ∈ CtlDef(t, v), s ∈ OmsCond(t, v)が定義される[1]．

（例）2つの値x, yの最小値，最大値を求めるプログラムmin_max（図1）に入力x = 3, y = 2を与えて実行した時の実行系列を図2に示す．t_I は実行時点tにおいて命令Iが実行されたことを表わす．実行時点6において，2つの変数min, maxの値が誤っている．実行時点6から変数maxに関してDef, CtlDef依存関係を辿ると，実行時点1, 4, 5が抽出される．これらの実行時点で実行された命令1, 4, 6は確かに実行時点6における変数maxの値に影響を与えている．一方，抽出されなかった命令2, 3, 5については，たとえ値誤りバグを含んでいても，実行時点6における変数maxの値には影響を与えないことが分かる．

表1　バグの例

バグの種類	正しい文	誤った文
値誤りバグ		
文記述誤りバグ	X := Y + Z;	X := Y - W;
	if X>Y then	if X < Z then
文記述過多バグ		X := Y + Z;
設定漏れバグ		
文記述漏れバグ	X := Y + Z;	
名前記述誤りバグ	X := Y + Z;	W := Y + Z;

```
1      get(x, y);
2      min := x;
3      max := x;
4, 5   if x < y then min := y;        ("x>y"が正しい)
6      else max := y; end if;
7      put(min, max);
```

図1　プログラムmin_max

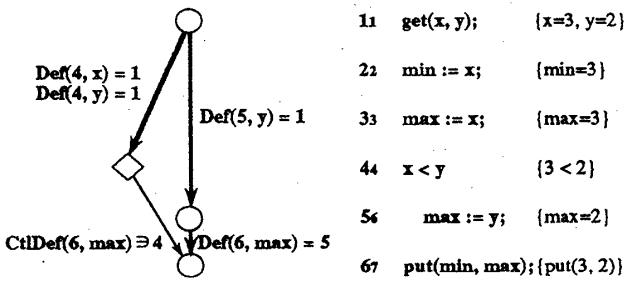

図2　変数maxへの影響

図2で示したように，依存関係を辿ることによって得られる有向グラフを**Critical-Flow**グラフと呼ぶ．実行時点jから依存関係を辿る場合には，その実行時点jにおいて使用されている各変数w ∈ Use(j)に対して，Def(j, w), CtlDef(j, w), OmsCond(j, w)となる実行時点を辿っていく．CriticalEP(t, v)は実行時点tから変数vに関してこれらの依存関係を辿ったときに得られるCritical-Flowグラフ内の実行

Optimizing Methods for Potential Fault Area *Critical Slice*
Takao Shimomura
NTT Software Laboratories
1-9-1 Kohnan, Minato-ku, Tokyo 108, Japan

時点の集合を表わす．CriticalEP(t, v) 内の要素を Critical 実行時点と呼ぶ．実行時点 t における変数 v に関する Critical Slice = CriticalSlice(t, v) は CriticalEP(t, v) 内の各実行時点で実行された命令の集合である．

3．プログラムバグ潜在域

プログラム P にある入力 I を与えて実行した時，実行時点 t における変数 v に関する変数値エラー E = VarValErr(t, v) に対して，次の条件を満たす，プログラム P 内の命令の部分集合 P' を，その変数値エラー E に関するプログラムバグ潜在域 と呼ぶ．

集合 P' に含まれない命令の値誤りバグは，変数値エラー E の原因とはならない．即ち，集合 P' に含まれない命令の値誤りバグを修正しても，変数値エラー E を修正することはできない．□

この定義によれば，もとのプログラム P 自身もプログラムバグ潜在域の 1 つになる．CriticalSlice(t, v) は自明でない 1 つのプログラムバグ潜在域を構成する．

4．プログラムバグ潜在域の最適化

実行時点 j において辿る対象となる変数の集合である Use(j) をできる限り小さい集合にとることによりプログラムバグ潜在域の最適化を実現する．変数 w ∈ Use(j) に関して実行時点 j から依存関係を辿ることによって抽出される実行時点の集合を CriticalFlow(j, w) と表わすこととする．分岐命令の実行時点 j において，以下のように Use(j) の最適化を行う．

(Opt1) 分岐命令の条件式の値を決定した項 Term に含まれる，使用された変数の集合（これを Use(Term) で表わす）を Use(j) とする．

(Opt2) 条件式の値を決定した項が複数存在する場合には，最終的に Critical-Flow グラフのサイズを最小とする項 $Term_i$ の $Use(Term_i)$ を Use(j) とする．

（例）図 3 に示すプログラム prog_cond に入力 x = 1, y = 2 を与えて実行した時の実行系列，および，プログラムバグ潜在域の最適化を行った結果を表 2 に示す．

(Opt0) Critical-Flow グラフから求められる Critical 実行時点を示す．

(Opt1) 実行時点 11 で実行された分岐命令の条件式 "x > 4 or z < 2" の値を決定した項 "z < 2" に含まれる変数 z のみを Use(11) として，最適化を行った結果である．

(Opt2) 実行時点 8 で実行された分岐命令 "x + 2 < 0 and y > 1" では，どちらの項も False となり条件式の値を決定している．CriticalFlow(8, x) = { 1, 2, 3, 5 }, CriticalFlow(8, y) = { 2, 3, 6 } である．そこで，最終的に Critical-Flow グラフのサイズを最小とする項 "y>1" を選択し，Use(8) = { y } として，最適化を行った結果である．

```
1       get(x);
2       get(y);
3       z := 1;
4       w := 1;
5       x := x + y + z -1;
6       y := y - z;
7       w := w + x;
8, 9    if x + 2 < 0 and y > 1 then z := 2;
10      else w := 3; end if;
11      x := w + 1;
12, 13  if x > 4 or z < 2 then w := 4; end if;
14      put(w);
```

図 3　プログラム prog_cond

表 2　プログラムバグ潜在域の最適化例

実行系列		Opt0	Opt1	Opt2
1_1 get(x);	{x = 1}	○	○	×
2_2 get(y);	{y = 2}	○	○	○
3_3 z := 1;	{z = 1}	○	○	○
4_4 w := 1;	{w = 1}	×	×	×
5_5 x := x + y + z -1;	{x = 3}	○	○	×
6_6 y := y - z;	{y = 1}	○	○	○
7_7 w := w + x;	{w = 2}	×	×	×
8_8 x + 2 < 0 and y > 1	{5 < 0 and 1 > 1}	○	○	○
9_{10} w := 3;	{w = 3}	○	×	×
10_{11} x := w + 1;	{x = 4}	○	×	×
11_{12} x > 4 or z < 2	{4 > 4 or 1 < 2}	○	○	○
12_{13} w := 4;	{w = 4}	○	○	○
13_{14} put(w);	{put(4)}	10	8	6

参考文献

1) 下村隆夫: 変数値エラーにおける Critical Sliceに基づくバグ究明戦略，情報処理学会論文誌，Vol. 33, No. 4, pp. 501-511 (1992).