ソフトウエア工学 23－2
（1982. 5. 7）

設定参照グラフを用いたプログラム誤り自動検出法

堀田 博文，永瀬 淳夫，細谷 僚一
（日本電信電話公社 横須賀電気通信研究所）

1．はじめに

プログラムの信頼性向上をねらいとして，データフロー解析を用いて翻訳時にプログラムの誤りを検出する方法が提案されている[1][2]。データフロー解析に基づく誤り検出法は，プログラムをフローグラフに変換し，そのフローグラフを解析することにより誤りを検出する方法であり，処理が簡潔で，かつ人手の介入を必要としない等の長所をもつ一方，以下の問題点を有していた。

(1) プログラム内の実行不能パス[4]が識別できないため，誤り検出が不正確である。

(2) プログラム内の変数状態（値の設定済，参照済等）しか解析できないため，検出可能な誤りが，変数の設定参照関係異常等，極く少数のものに限られる。

筆者らは，変数値域を静的に追跡する方法を採用することにより，データフロー解析の欠点を解決する誤り検出法を考案した。本論文では，まず，プログラム内の変数値域を制御の流れに沿って静的に追跡することにより，実行不能パスの識別や配列の添字範囲逸脱，無限ループ等の検出が可能となり，(1)(2)の両問題が解決されることを示す。次に，変数値域に基づく誤り検出を効率よく実現するために，「設定参照グラフ」という概念を導入し，これを利用した誤り検出方式を提案する。

2．従来の誤り検出法の問題点

データフロー解析[3]に基づく誤り検出は，次の2段階で行われる。①プログラムの制御構造の抽象表現であるフローグラフに沿って，変数の状態（値の設定済，参照済等）を伝播させる。②その結果得られたプログラム内各地点における変数の状態を用いて，変数の設定参照関係異常（データフローアノマリ）を検出する。

この方法は，フローグラフのみの解析によりプログラム内各地点での変数の状態を求めるため，処理が簡潔で効率がよく，人手の介入も不要であるという長所をもっている。しかし，誤り検出能力の面で，次のような問題がある。

(1) 誤り検出が不正確である。

(2) 検出可能な誤りの種類が限られている。

これらの問題点について，以下に詳述する。

(1) データフロー解析は，フローグラフ内のすべてのパスが実行されるものと仮定して，変数の状態を解析する。ところが，フローグラフには，プログラム上では実際には起こり得ない制御執行を表わすパス（「実行不能パス」と呼ぶ[4]）も存在する。このため，データフロー解析に基づく誤り検出法では，実際には起こり得ない動作（実行不能パス上で生じる動作）がプログラム内に存在するものとみなし，次のような不正確な誤り検出を行う場合がある。

(i) 起こり得ない「未設定変数の参照」を異常として検出する。

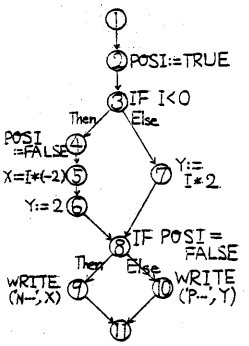
(ii) 「有効でない値の設定」（冗長な代入文）を見逃す。

これらの例を以下に示す。

図1(a)のプログラム*には，図1(b)のフローグラフが対応する。このフローグラフにデータフロー解析を適用すると，(I)「⑨で値が未設定の変数Xを参照している」という異常が検出される。この異常は，パス(①,②,③,⑦,⑧,⑨,⑪)が実行されたときに起こるものであるが，このパスは実行不能パスであり，上記の異常は実際には起こり得ない，すなわち，不適切な誤り検出を行ってしまう((i)の例)。また，(II)「⑥で設定された変数Yの値は⑩で参照される」という解析結果が得られ，⑥の代入文は有効であるとみなされる。しかし，⑥と⑩を通るパス(①,②,③,④,⑤,⑥,⑧,⑩,⑪)は実行不能パスであり，⑥で設定されたYが⑩で参照されることは実際にはあり得ない。すなわち，⑥でのYの設定は，以後参照されない冗長なものである((ii)の例)。

```
PROCEDURE P(I:INTEGER);
  VAR POSI : BOOLEAN;
      X,Y  : INTEGER;
① BEGIN
②   POSI:=TRUE;
③   IF I<0
      THEN BEGIN
④              POSI:=FALSE;
⑤              X:=I*(-2);
⑥              Y:=2
             END
⑦     ELSE Y:=I*2;
⑧   IF POSI=FALSE
⑨     THEN WRITE('NEGATIVE',X)
⑩     ELSE WRITE('POSITIVE',Y)
⑪ END;
```

(a) プログラム例



(b) (a)のフローグラフ

図1. データフロー解析例

本問題の解決策として，プログラム内の実行不能パスを検出することにより，誤り指摘メッセージの中から不適切な(実行不能パス上の)ものを除去する方法が提案されている[4]。しかし，この方法では，(i)の問題点には対処できるが(ii)には対処できないという問題が残されている。

(2) データフロー解析では，フローグラフに沿って変数の状態のみを伝播させるため，検出可能な誤りは，変数の設定参照関係異常に限定される。すなわち，
(i) 配列の添字範囲逸脱，ゼロ除算等の変数値域に関連した誤り
(ii) 実行不能文，無限ループ等の制御飛行に関連した誤り
等は，検出できない。

これまでに，誤り検出項目の拡大をねらいとして，ポインタ変数の使用に関連する誤りを検出する方法[5][6]や，状態遷移図で示される操作系列の規則**にプログラム動作が従っていることを検証する方法[7]が，提案されている。しかし，上記(i)(ii)の誤りを含めて，網羅的かつ統一的にプログラム誤りを検出する方法は，これまで提案されていない。

3. 変数値域を利用した誤り検出

3.1 変数値域による実行不能パス識別

第1の問題点を解決するためには，プログラム内の実行不能パスを識別する必要がある。その一手法として，各パスが実行される条件を求め，それが実行可能か否かを検査するものがある[4]。この方法は，論理式を評価するため，実行不能パスの識別を高精度に行うことができるが，処理が複雑で時間が多くかかるという欠点をもつ。本論文では，処理の簡単さや処理効率の向上をねらい

* 以後，プログラム例は，PASCAL記述とする。

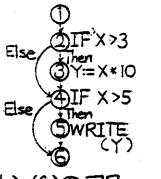
** ファイルの操作は，open実行後でなければならない等の，処理の順序に関する規則。

として，変数値域を利用して実行不能パスを識別する方法を提案する。

これは，プログラム内のある地点が実行される条件を変数の値域で代用し，条件の排反を値域の共通部分の有無で検査するものである。たとえば，図2の例では，②→④，④→⑤の制御移行が生じるときのXの値域は，それぞれ，[−∞, 3]，[6, ∞]である。この両地点のXの値域には，共通部分はなく（[−∞, 3] ∩ [6, ∞] ≡ φ*），さらに，両地点間には，Xへの値の設定は存在しない。この解析結果から，パス（①, ②, ④, ⑤, ⑥）が実行されるようなXの値は存在せず，このパスは実行不能であることがわかる。このようにして，変数値域の利用により，パスの実行不能性が認識できる。

```
PROCEDURE Q(X:INTEGER);
  VAR Y : INTEGER;
① BEGIN
②   IF X>3
③     THEN Y:=X*10;
④   IF X>5
⑤     THEN WRITE(Y)
⑥ END;
```

(a) プログラム例



(b) (a)のフローグラフ

図2. 実行不能パスを含むプログラム例

3.2 変数値域による誤り検出

変数値域の認識は，第2の問題点の解決にもつながる。すなわち，変数値域に関連した誤りを検出でき，さらに，制御移行に関連した誤りも検出可能となる。たとえば，図3のフローグラフでは，④を通るパスは（①, ③, ④, ⑥）のみであり，しかもこのパスは実行不能パスであるため，④は実行不能文であることが検出できる。また，図4のフローグラフでは，ループを脱出するパス（①, ②, ③, ④）は実行不能パスであるため，このループは実行不能パスであることが検出できる。

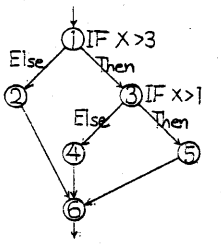


図3. 実行不能文の例

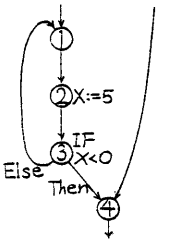


図4. 無限ループの例

3.3 設定参照グラフを用いたプログラム解析手法

前述のように変数値域を導入して誤りを検出する場合，必要とされる基礎的な情報は，

(1) 変数の設定参照関係**
(2) 変数値域
(3) パスの実行不能性

であり，これらの情報の間には，次のような密接な依存関係がある（図5参照）。

(i) 変数値域を求めるためには，変数の設定参照関係が必要である。
(ii) 変数値域を認識することにより，パスの実行不能性が認識できる。
(iii) 実行不能パスを用いて，変数の設定参照関係をより正確に認識する。

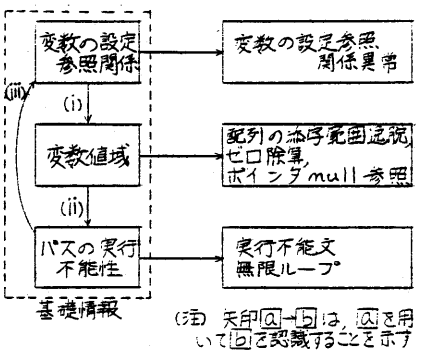


図5. 本技法における情報の流れ

---

* ∩は，値域を値の集合とみなした時の積集合の意味である。

** プログラム内のどの地点で設定された変数値をどこで参照しているかという関係。

また，これらの情報は，それぞれ，図5の矢印の順に解析を繰返すことにより，より正確なものとなる。したがって，(1)(2)(3)の情報およびその関連を効率よく管理する必要がある。従来のデータフロー解析で用いるフローグラフは，パスの実行条件を意識せずに解析を行うための概念であるため，この条件を満足しない。

本技法では，各情報が密に関連するため，それらを一元管理する必要がある。これらの情報は，すべて，(1)の変数の設定参照関係から求められ，さらに，変数の設定参照関係とは，プログラム上の2地点の関係であることを考慮すれば，本技法で行う情報管理は，2地点間の関係のみを表現できればよいことが判明する。

そこで，本論文では，①解析の繰返しによる情報変化を簡潔な方法で把握でき，しかも，②3つの情報の一元管理が可能であることをねらいとして，次のように定義される設定参照グラフを導入する。

【定義】

設定参照グラフは，次の性質をもつ有向グラフである。

(i) ノードは，プログラムの文に相当する。さらに，プログラムの開始，終了に相当する2つのノードが存在する。

(ii) 文aで設定された変数値を文bで参照しているとき，文aを示すノードから文bを示すノードへのエッジが存在する。ただし，プログラムの開始を示すノードでは，すべての変数に値が設定されているものとみなす。■

ここで，変数の設定及び参照は，IF文等の条件分岐を除き，従来のデータフロー解析と同様である。なお，条件分岐では，条件式に出現する変数が，まず参照され，その分岐先毎に値が設定されているとみなす。

図1のプログラムに対応する設定参照グラフを，図6に示す。

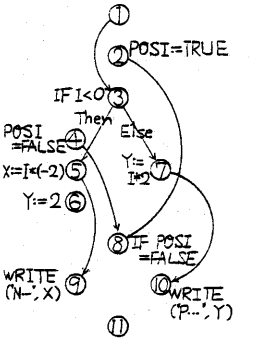


図6. 図1の設定参照グラフ

本手法をとることにより，次のように従来技法の問題点が解決される。

① 実在する設定参照関係は，設定参照グラフ上のエッジの有無により表現，把握できるため，実行不能パスを意識した正確な誤り検出が容易に行える。

② 変数の設定参照関係異常の検出に必要な情報のみでなく，ノードに変数値域を付随させることにより，値域決定や実行不能パス識別に必要な情報も設定参照グラフ上に表現される。そのため，このグラフのみを用いて，種々の誤り検出に必要な情報を一元的に管理でき，多くの種類の誤り検出が容易となる。

4. 誤り検出アルゴリズム

本技法では，設定参照グラフを用いて，図5の矢印に沿って解析を繰返し，必要な情報を求め，誤り検出を行う。その手順を以下に示す（図7参照）。

step1 データフロー解析に基づき，設定参照グラフを求める。

step2 情報が収束するまで，以下の処理を繰返す。

step2.1 設定参照グラフを用いて，変数値域を決定する。

step2.2 変数値域を用いて実行不能パスを識別する.

step2.3 実行不能パスの情報を用いて，設定参照グラフ上から，起こり得ない設定参照関係を示すエッジを除去する.

step3 step2で求められた情報を用いて，誤りを検出する.

以下では，まず，本章で用いる記法の定義を述べ，次に，手続き内の解析方法，次いで，手続き間への解析の拡張方法について示す.

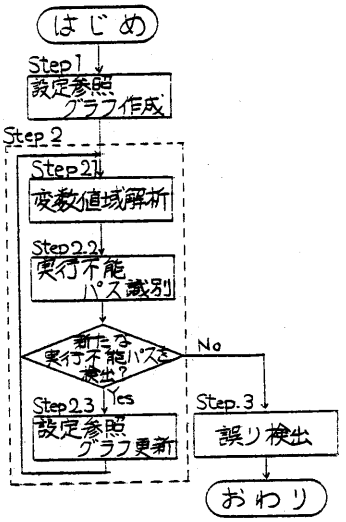


図7. 誤り検出アルゴリズム

4.1 記法の定義

(1) 設定参照グラフを $G_{DR}=(N_{DR}, E_{DR}, s)$ で表わす．ただし，N_{DR} はノード集合，E_{DR} はエッジ集合，s は開始ノードを示す.

(2) 文をノードとするフローグラフを G_F とする.

(3) ノード i に対応する文を実行する直前，直後を $\to i$，$i \to$ で表現する．すなわち，$\to i$ は，ノード i に入るエッジの集合に相当し，$i \to$ は，ノード i から出るエッジに相当する．なお，i がIF文の場合は*，実行後の分岐先が2箇所あるため，そのThen側，Else側に対応させて，それぞれ $i_T \to$，$i_E \to$ で i の実行直後を表現する.

(4) $D(\to i, v)$，$D(i \to, v)$ で，それぞれ，ノード i の実行直前，直後における変数 v の値を設定した地点の集合（設定地点集合）を表わす.

(5) $R(\to i, v)$，$R(i \to, v)$ で，それぞれ，ノード i の実行直前，直後における変数 v の値域を表わす.

(6) $a \nrightarrow b$ で，地点 a の実行後に地点 b が実行され得ないことを示す.

(7) $Dominated(e) = \{$ノード $m \mid G_F$ 上，ノード m を通るパスは必ず e を通る $\}$[3]。

4.2 手続き内の解析

(1) 設定参照グラフの作成

まず，データフロー解析を用いて，実行不能パスを意識せずに，変数の設定参照関係を求める．このとき，G_F 上を伝播させるものは，「各変数の設定地点情報」である．本解析の結果から，プログラム内各地点での各変数の設定地点集合Dを求め，これを用いて設定参照グラフ G_{DR} を求める.

(2) 変数値域の解析

変数値域を解析する基本的な手法は，文献[8]で提案されている．しかし，この方法は，データフロー解析で求めた設定地点集合を用いるため，認識される値域は，実行不能パスを意識しないものとなっている．これに対して，本技法では，論理上起こり得ない設定参照関係を除いた設定地点集合Dに基づいて作成された設定参照グラフ G_{DR} を使用するため，より正確な値域が求められる．解析方法の概略を以下に示す.

---

* CASE文等，IF文以外の条件分岐は，IF文の組合せで表現できることを考慮して，ここでは条件分岐としてIF文のみを考えることにする.

プログラム内のある地点nで，変数$v_1, v_2, \cdots, v_l$の値を参照して変数v_0に値を設定しているとする。このとき，変数v_k ($k=1,2,\cdots,l$)の地点nにおける値を設定した地点$D(\rightarrow n, v_k)$は，$(i, n) \in E_{DR}$であるノードiの集合によりG_{DR}上に表現されている。この情報を用いれば，$R(\rightarrow n, v_k)$は，次の式で求められる。

$$R(\rightarrow n, v_k) = \bigcup_{i \in D(\rightarrow n, v_k)} R(i \rightarrow, v_k)$$

なお，Uは，値域を値の集合とみなした場合の和集合を表わす。さらに，nにおいて値が設定される変数v_0の値域$R(n \rightarrow, v_0)$は，$R(\rightarrow n, v_k)$を用いて認識できる。その方法については，文献[8]を参照されたい。G_{DR}上で値域を決定する例を図8に示す。なお，図8では$\rightarrow n$, $n \rightarrow$における値域を，それぞれノードnの上方，下方に示した。

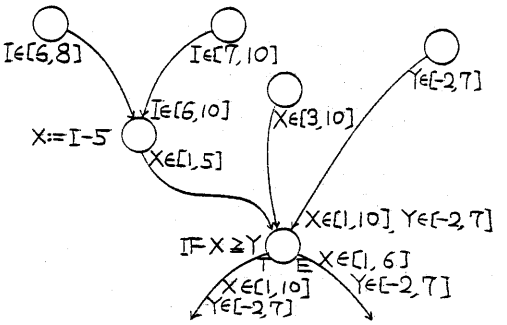


図8. 設定参照グラフ上での値域決定例

(3) 実行不能パスの識別

変数の設定地点と参照地点の関係は，プログラム上の2地点の関係であることに着目し，本技法では，プログラム上の2文（G_{DR}上の2ノード）間の実行条件排反を，認識された変数値域を用いて検査し，変数の設定参照関係をより正確に把握する方法をとる。

$a \rightarrow$に対応する制御移行が行われた後，$b \rightarrow$に対応する制御移行が起こり得ない（$a \rightarrow \not\rightarrow b \rightarrow$である）条件は，以下の通りである。

(i) bは，IF文に対応する。

(ii) ある変数vに対して，以下の条件が成立する。

① $R(a \rightarrow, v) \cap R(b \rightarrow, v) \equiv \phi$

② $a \in D(\rightarrow b, v)$

③ G_F上でaからbに至るどのパスでも，vは，a，b以外の地点で値の設定をうけない*

上記の検査により，$a \rightarrow \not\rightarrow b \rightarrow$が判明すれば，$x \in Dominated(a \rightarrow)$, $y \in Dominated(b \rightarrow)$である任意のノードx，yに対し，$x \not\rightarrow y$が成立する。

(4) 設定参照グラフの更新

(3)の方法で実行条件が排反するノード対を求め，その結果を利用して設定参照グラフG_{DR}を更新する。すなわち，

(i) 実行条件が排反するノード間の設定参照関係がG_{DR}に登録されていれば，その関係を示すエッジをE_{DR}から除去する。

(ii) 未設定参照の関係を示すE_{DR}上のエッジ(s, i)のうち，実行不能パス上でのみ生じているものをE_{DR}から除去する。

という処理により，より正確な設定参照関係を示すG_{DR}を求める。

本技法は，図7に示すように，上述のG_{DR}更新操作を繰返す。この繰返し

---

* vの設定地点集合を検査することにより，本条件の成否がチェックできる。そのアルゴリズムの詳細については，文献[8]を参照されたい。

は，G_{DR}のエッジ集合E_{DR}の大きさを単調減少させるのみであるため，有限回の繰返しで情報は収束する。

(5) 誤りの検出

G_{DR}上のエッジの有無より変数の設定参照関係異常を，変数値域より値域関連の異常を，実行不能パス（具体的には，実行条件が排反する2地点）より実行不能文や無限ループを，検出する。その詳細を表1に示す。

表1．誤り検出方法

誤りの種類	検出方法
変数の設定参照関係異常	設定参照グラフ内に，後続ノードを持たない代入文ノードがあれば，それは，以後参照されない値の設定（冗長な代入文）を示している。また，プログラムの開始に相当するノードを先行ノードとしてもつ参照地点は，値が未設定である変数を参照している。
ポインタのnull参照	値域がnullであるポインタ型変数を修飾子として用いれば，それはアクセス誤りである。
配列の添字範囲逸脱	配列要素参照時，添字の値域が宣言値域内になければ，配列の領域外アクセスの危険性がある。
ゼロ除算	値域が0の変数（または式）で除算すれば，それはゼロ除算である。
実行不能文	ノードaを通るパスがすべてノードbを通る（Dominated(b) ∋ a である）とき，aとbの実行条件が排反するならば，ノードaは実行不能文を示している。
無限ループ	プログラムループの入口ノードaとそのループ脱出直後に実行されるノードbの実行条件が排反するならば，そのループは無限ループである。

4.2 手続き間の解析

手続き間にわたる解析の方法には，代表的なものとして，以下の2つがある。

ⓐ 手続き呼出しをインライン展開して解析する。

ⓑ 個々の手続きを別個に解析し，そのインタフェースをチェックする。

このうち，ⓐは，解析時間やメモリ量が膨大となり，また，再帰呼出しを解析できないため，ここでは，ⓑの方法を採用する。このとき，手続き間インタフェースは，呼出される手続きの解析結果を用いて呼出し側の手続きを解析することにより，チェックすることとした。

したがって，手続きの解析は，呼出し関係の下位から上位への順(r-invocation order[3])で行う。なお，再帰呼出しを含む場合は，再帰呼出しを行う手続きの集合をまず求め，次に，各手続きに対して，解析可能部分（未解析手続きの呼出しを含まない部分）の解析を，すべての手続き内に未解析部分がなくなるまで繰返す。手続き間インタフェースは，以下のように解析する。

変数の設定参照関係については，手続きPを解析した結果，引数や大域的変数vに関して，① $(s, i) \in E_{DR}$ なるノードiがあれば，各手続きのPの呼出し地点では，vが参照されている，②終了ノードnに対して，$D(\rightarrow n, v) \ni \{s\}$

であれば，各手続きのPの呼出し地点では，ひが設定されている，とみなして，Pを呼出している手続きの解析を行う。

また，変数値域については，手続きPを呼出している手続きの解析時に，Pの出口での引数や大域的変数ひの値域（解析済み）が，Pの呼出し地点で設定される変数ひの値域であるとみなす。

5. おわりに

本論文では，従来のデータフロー解析に基づくプログラム誤り検出法が有していた以下の問題点を解決する方法について述べた。

(1) 誤り検出が不正確である。
(2) 検出可能な誤りの種類が少ない。

本技法の方式上の特徴は，以下の通りである。

ⓐ プログラム内の変数値域を認識する。
ⓑ 変数値域を用いて実行不能パスを識別する。
ⓒ 解析に必要な情報を，すべて設定参照グラフ上で一元管理している。

また，本技法は，従来技法に比べ，以下の点で誤り検出能力及び精度を向上させている。

(a) 実行不能パス上の異常を指摘する不適切なメッセージの削減や，従来技法では見落とされていた冗長な代入文の検出が可能である。
(b) 変数値域に関連した誤り（配列の添字範囲逸脱，ゼロ除算，ポインタのnull参照）や，制御飛行に関連した誤り（実行不能文，無限ループ）が検出可能である。

今後は，本技法を，Ada 等の良形な制御構造や高水準のデータ構造をもつ言語を対象として実現し，その効率（処理時間，メモリ量）や誤り検出能力の評価を行う予定である。

参考文献

[1] Osterweil,L.J. & Fosdick,L.D.,"DAVE-A Validation Error Detection and Documentation System for FORTRAN Programs," Software-Practice and Experience, vol.6 (1976)

[2] 宮本,"PASCALプログラムにおける変数定義・使用に関するデータフロー解析," 情処論, vol. 21, No.1 (1980)

[3] Hecht,M.S.,"Flow Analysis of Computer Programs," North-Holland (1977)

[4] Osterweil,L.J.,"The Detection of Unexecutable Program Paths through Static Data Flow Analysis," Proceedings on COMSAC 77 (1977)

[5] Cousot,P. & Cousot,R.,"Static Determination of Dynamic Properties of Generalized Type Unions," Proceedings of an ACM Conference on Language Design for Reliable Software (1977)

[6] 岡 他，"ポインタ変数に関するプログラム誤りについて，" 情処学会 ソフトウェア工学研究会資料 21 (1981)

[7] 花田 他 "データフロー解析を応用したオペレーション実行系列の検証，" 信学論, Vol. J64-D, No.12 (1981)

[8] Harrison,W.H.,"Compiler Analysis of the Value Ranges for Variables," IEEE Transactions on Software Engineering, vol.SE-3, No.3 (1977)