論　文

広域データフロー解析に基づく関数型プログラムの変則性検出

正　員　小川　瑞史†　　正　員　小野　諭†

Anomaly Detection of Functional Programs Based on Global Dataflow Analysis

Mizuhito OGAWA † *and* Satoshi ONO †, *Members*

あらまし　一階の関数型言語を対象として，広域データフロー解析に基づく新たな変則性検出アルゴリズムを提案する．変則性検出は，評価しても結果に影響を与えることのない不要オブジェクト，および評価すると無限ループに陥り停止しない発散オブジェクトの2点について行う．この検出アルゴリズムは，計算結果を与える計算経路を求める関数間広域解析技術に基づいている．このため，単に構文的な定義・参照関係を追跡するクロスレファレンス型解析より高い検出能力をもっている．本アルゴリズムを用いると，概念的には無限長リストとなるストリームを用いたプログラムにおいて生じやすい，計算が発散するバグの自動検出に有効である．

1. まえがき

関数型言語は副作用を排除した宣言型言語の一つである．関数型言語の特徴の一つとして，部分的な入力により計算の実行を可能とするノンストリクトネス[10]が挙げられる．例えば，概念的には無限長となるリスト構造であるストリームを実現することができ，これは独立した複数の処理相互でデータを受け渡し同期を保つ方法として，よく使用されている．

ストリームを用いた計算では，結果に必要な部分のみ評価を行う要求呼び(Call by Need)計算[1]が必要である．要求呼び計算は，参照している引数すべてを先に評価する値呼び(Call by Value)計算[2]に比べ，一般に効率が悪いため，計算が発散しない限り，要求呼びを値呼びに置き換えてプログラムする手法が用いられる．この置換えはコンパイラにより完全自動化することが望ましく，データ構造のない場合については，いくつかのアルゴリズムが提案されている[6],[7],[12],[14]．しかし，ストリームなどの再帰的データ構造をもつ場合の自動変換の手法については，まだ明らかにされていない．

データ生成時にその要素を評価する(値呼び)か，遅延するか(要求呼び)かをプログラマに指定させる場合においては，本来，要求呼びにすべきところを値呼びにしてしまい，計算が発散してしまうバグが生じやすい．このような誤りに対しては，広域解析を用いて，バグを生じる恐れのあるデータの依存関係の異常を検出する変則性検出(Anomaly Detection)が有効である[16]．

プログラムの変則性とは，データ依存関係に異常があり，エラーを招いたり冗長な計算を行う可能性のある表現をいう．関数型言語においては，

- 定義されたが参照されない未参照オブジェクト，
- 定義なしに参照された未定義オブジェクト，
- 参照されるがその値が結果に影響を与えない不要オブジェクト，
- 計算が出口のない無限再帰に陥り停止しない発散オブジェクト，

などが対応する(但し，オブジェクトとは関数，関数引数，局所変数などを指す)．

従来，手続き型言語においては，データの依存関係を解析する広域解析[2]を用いて，変則性を検出するアルゴリズムの研究がなされてきた[4],[5]．これらは，手続きごとに独立に解析を行っていたため，再帰的に定義されたオブジェクトを扱う関数型言語に対しては，特にそのノンストリクス性に関し，十分な解析ができなかった．また，プログラムの構文的な定義参照関係を

† NTT ソフトウェア研究所，武蔵野市
NTT Software Laboratories, Musashino-shi, 180 Japan

追跡するクロスリファレンス型解析では，未定義/未参照オブジェクトは検出できるものの，不要/発散オブジェクトの検出はできなかった．

本論文では一階の関数型プログラムを対象として，関数間全域にわたる広域データフロー解析を用いた新たな変則性検出アルゴリズムを提案する．

不要/発散オブジェクトの検出は，関数の引数・結果の関数において生じる外部変則性（External anomaly）と，一つの関数本体内の計算において生じる内部変則性（Internal anomaly）の二つのルベルで行う．検出の対象は，外部変則性として，不要な関数引数および発散する関数定義，内部変則性として，不要な局所変数および発散する局所変数定義である．

変則性検出の基礎となる解析として，外部変則性に対応する関数間レベルでは，プログラム全体を同時に解析する関数間解析である引数依存属性集合（Parameter Dependency Property Set：以下，PDPS と略す）解析[10)~12),15)]を用いる．また，内部変則性に対応する関数内レベルでは，PDPS 解析結果に基づき関数本体内の計算経路を再現する部分値依存属性集合（Partial Resulting-Value Dependency Property Set：以下，VDPS と略す）解析[12),14)]を用いる．そしてそれぞれのレベルにおいて，各計算経路で参照されることのない不要オブジェクトの検出，計算経路が関数間ループまたは関数内ループとなる発散オブジェクトの検出を行う．

本論文の構成は次のとおりである．**2.** では関数型言語のノンストリクトネスおよび計算経路解析など基礎となる概念について説明する．**3.** では提案するアルゴリズムの概要について述べる．そして，**4.** では提案するアルゴリズムの能力について論ずる．

2. 関数型プログラムのノンストリクトネス

2.1 データ構造のない領域上のノンストリクトネス

ノンストリクトネスとは入力の一部が未定義でも計算結果が求まるプログラムの性質をいう．また，関数 $f(x_1, x_2, \cdots, x_n)$ を評価するためにつねにすべての引数の評価が必要となるときストリクト関数，そうでないときノンストリクト関数という．例えば，$x+y$ は，和を求めるためにすべての引数の評価が必要なためストリクト関数であり，$\mathrm{if}(x, y, z)$ は x が true ならば y のみ，また x が false ならば z のみの評価で結果が得られるためノンストリクト関数である．ユーザ定義関数についても同様で，アッカーマン関数

$$\begin{aligned}\mathrm{ack}(x, y) = \mathrm{if}\ \ & x==0 \ \ \mathrm{then}\ \ y+1\\ & \mathrm{elseif}\ \ y=0 \ \ \mathrm{than}\ \ \mathrm{ack}(x-1, 1)\\ & \mathrm{else}\ \ \mathrm{ack}(x-1, \mathrm{ack}(x, y-1))\end{aligned}$$

はストリクト関数であり，タライ関数

$$\begin{aligned}\mathrm{tarai}(x, y, z) = \mathrm{if}\ \ & x \geqq y\\ \mathrm{then}\ \ & \mathrm{tarai}(\mathrm{tarai}(x-1, y, z),\\ & \quad \mathrm{tarai}(y-1, z, x),\\ & \quad \mathrm{tarai}(z-1, x, y))\\ \mathrm{else}\ \ & y\end{aligned}$$

は，$(x, y, z)=(2, 3, z)$ のように，z の値によらず結果が定まる場合があるので，ノンストリクト関数である．ノンストリクトネスは，要求駆動型計算において，関数 $f(x_1, x_2, \cdots, x_n)$ に評価要求が生じたときの各引数 x_i への評価要求伝搬のパターンとして表現される．評価要求の伝搬のパターンは，データ構造のない領域上では，要求の伝搬する引数の組からなる集合により表すことができる．例えば，関数 $x+y$ においては $\{\{x, y\}\}$ である．また，関数 $\mathrm{if}(x, y, z)$ においては $\{\{x, y\}, \{x, z\}\}$ であり，それぞれ，x が true・false の場合に対応している．

関数 $f(x_1, \cdots, x_n)$ に対し，このような引数の組を極小充足引数集合（Minimally Sufficient Parameter Set：以下，MSPS と略す）[10),13)]，また MSPS 全体からなる集合を，PDPS[10),13)] と呼び，$\mathcal{D}(f(x_1, \cdots, x_n))$ と表す．例えば，$\mathcal{D}(x+y)=\{\{x, y\}\}$，$\mathcal{D}(\mathrm{if}(x, y, z))=\{\{x, y\}, \{x, z\}\}$ となる．

2.2 データ構造がある領域上のノンストリクトネス

データ構造のない領域においては，データは未評価であるか評価済みのいずれかである．しかしデータ構造のある領域においては，データコンストラクタに与えられた意味によって，部分的に評価されたデータが生じる場合がある．例えば，Common-Lisp における cons は，データ構造を生成する前に car 部・cdr 部ともに評価するが，Scheme[3] における cons-stream は cdr 部を遅延評価[1] するコンストラクタであり，cdr 部を未評価のままリストを生成する．同様に，car 部・cdr 部ともに遅延評価するコンストラクタ lazy-cons は，car 部・cdr 部ともに未評価のままリストを生成する．cons のようなストリクトなコントラクタをストリクトデータコンストラクタ，そうでない cons-stream，lazy-cons などをノンストリクトデータコンストラクタと呼ぶ．ノンストリクトデータコンストラクタを用いると，

$$\mathrm{intseq}(n) = \mathrm{cons\text{-}stream}(n, \mathrm{intseq}(n+1))$$

のように概念的には無限長となるデータ構造（ストリー

ム）が構成できる．

データ構造がある領域上の関数においては，intseq(n)の定義本体内のcons-streamのように，関数に評価要求が伝搬しても，引数部の評価を必要になるまで遅延させる場合がある．従って，関数のノンストリクトネスを表現するためには，各MSPS内の各引数に対し変数参照モード[12]が必要となる．関数に評価要求が生じたとき，各評価要求パターン（MSPS）において，評価を遅延される引数をDモード（Delayed Mode），評価される引数をSモード（Strict Mode）と呼び，このモードにより，それぞれの評価要求の伝搬のパターンを表現する．以下，このようなモード付きPDPS[12]を，単にPDPSと呼び，各MSPSに属するDモードの引数のみ上付バーのタグをつけて表す．例えば，上記cons(x, y)，cons-stream(x, y)，lazy-cons(x, y)のPDPSは，それぞれ，$\{\{x, y\}\}$，$\{\{x, \bar{y}\}\}$，$\{\{\bar{x}, \bar{y}\}\}$である．データ構造のない領域上の関数では，MSPS内の各引数の参照はすべてSモードである．ユーザ定義関数についても同様に，

$f_1(x, y)$=if x==0 then cons(x, y)
else cons-stream(x, y)

に対し，$\mathcal{D}(f_1(x, y))=\{\{x, y\}, \{x, \bar{y}\}\}$などとなる．

2.3 関数間計算経路解析（引数依存属性集合解析）

既に与えられた基本関数のPDPSをもとに，ユーザ定義関数のPDPSを導く手法がPDPS解析[10),12),13]である．PDPS解析は関数の適用により伝搬する性質を求めるため，プログラム全体を同時に解析する関数間広域解析となっている．

そのアルゴリズムを簡単に説明すると，各ユーザ定義関数に対するPDPSの第1次近似をϕとする．そして，基本関数のPDPSとユーザ定義関数の第n次近似のPDPSをもとに，ユーザ定義関数のPDPSの第(n+1)次近似を計算する．この操作をすべてのユーザ定義関数のPDPSが収束するまで繰り返す．詳しくは文献[10), 12), 13)]を参照されたい．

PDPS解析の特徴は次の三点である．

（1） MSPSを関数の計算経路上参照する引数の組とみなしたとき，PDPS解析は再帰からある出口に到達する計算経路のみ検出する．従ってすべての計算経路が無限再帰に陥り計算が発散する関数をPDPSが空集合になることを用いて検出できる．例えば，

intseq(n)=cons-stream(n, intseq(n+1))
intseq′(n)=cons(n, intseq′(n+1))
diverged(x, y)=if $x+y<0$
then diverged($x+1, y$)
else diverged($x-1, y$)

に対し$\mathcal{D}$(intseq′(n))=$\mathcal{D}$(diverged(x, y))=ϕとなり，intseq′(n)，diverged(x, y)は，ともに発散することがわかる．これに対し，$\mathcal{D}$(intseq(n))=$\{\{n\}\}$となり，intseq(n)は，結果を導きうる計算経路をもつことがわかる．

（2） MSPS全体の交わりをとることにより必須引数（Strict Parameter：関数を評価する上で必ず評価しなければいけない引数）[8),9)]が得られ，結びをとることにより有用引数（Relevant Parameter：関数を評価する上で評価が必要な場合がある引数）[10)]が得られる．例えば，

easy(x, y)=if $x=0$ then 1 else
easy($x-1$, easy(y, x))
$f_2(x, y)$ =if $x=0$ then 1 else $f_2(x-1, y+1)*y$

において，PDPSはそれぞれ$\mathcal{D}$(easy(x, y))=$\{\{x\}\}$，$\mathcal{D}(f_2(x, y))=\{\{x\}, \{x, y\}\}$となり，必須引数はともに$x$であり，有用引数はそれぞれ$\{x\}$，$\{x, y\}$であることがわかる．すなわち，PDPS解析は，ストリクトネス解析[8),9)]の上位の解析といえる．

（3） 各ユーザ定義関数のPDPS解析結果を用いて各関数本体内の各部分式の計算経路を再現することができる．その手法が次節で紹介する関数内解析であるVDPS解析[12),14)]である．PDPS解析とVDPS解析を合わせて計算経路解析（Computation Path Analysis）[10)]と呼ぶ．

2.4 関数内計算経路解析（部分値依存属性集合解析）

関数本体内の部分式expの計算経路は，関数本体内の各関数の出力ごとに変数名を割り当てるラベル付け変換（Labeling Transformation）を事前に行うことにより，式expを評価する上で必要となりうる変数（局所変数を含む）の組により表すことができる．

この組を極小充足部分値集合（Minimally Sufficient Value Set：以下，MSVSと略す），MSVS全体からなる集合をVDPSと呼び，$\mathcal{V}$(exp)と表す（但し，局所変数aのVDPSはaの局所定義本体のVDPSとする）．VDPS解析においてもPDPS解析と同様に，Sモード，Dモードの参照モードを用いる．各MSVSに含まれるDモードの変数は，PDPS解析と同様に上付きバーをタグをつけることにより表す．

関数本体内の各部分式のVDPSを，PDPS解析結果をもとに求める解析がVDPS解析である．例えば，

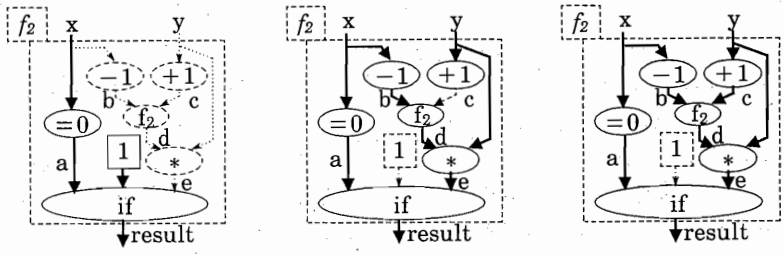

図1 関数 $f_2(x, y)$ の MSVS とデータフローグラフとの関係
Fig. 1 Relation between MSVSs of function $f_2(x, y)$ and dataflow graphs.

前節で示した関数

$f_2(x, y)$=if $x=0$ then 1 else $f_2(x-1, y+1)*y$

において PDPS 解析の結果は $\mathcal{D}(f_2(x, y))=\{\{x\}, \{x, y\}\}$ となる．これに対し $f_2(x, y)$ 内の各関数ノードの出力ごとに変数名（ラベル変数と呼ぶ）を割当て関数本体をネストしない letrec 式[1]で表すラベル付け変換を施すと，

$f_2(x, y)$={letrec $a=x==0$；$b=x-1$；$c=y+1$；
$d=f_2(b, c)$；$e=d*y$；
result=if a then 1 else e；
in result}

となる．このとき，関数 $f_2(x)$ の出力 result の VDPS $\mathcal{V}$(result) は $\{\{x, a\}, \{x, y, a, b, d, e\}, \{x, y, a, b, c, d, e\}\}$ となる．それぞれの MSVS は図1のように PDPS 解析結果 $\mathcal{D}(f_2(x, y))=\{\{x\}, \{x, y\}\}$ により表される各ノードの分岐を固定したそれぞれの計算経路（太線で表示された局所変数 result にいたる計算経路）に対応している．VDPS 解析のアルゴリズムは，PDPS 解析結果により表されるデータの依存関係に従い，関数本体内の各部分式の依存する変数の集合の推移的閉包（Transitive Closure）をとることに相当する．詳しくは文献(12)，(14)を参照されたい．

3. 変則性検出アルゴリズムの概要

3.1 変則性検出アルゴリズムの基本的考え方

プログラムの変則性とは，データ依存関係の異常をいい，その原因により，以下の4種類に分類される．

・定義がないのに参照される未定義オブジェクト．
・参照されないのに定義がある未参照オブジェクト．
・参照されるがその値が結果に影響を与えない不要オブジェクト．
・評価すると無限ループに陥り停止しない発散オブジェクト．

また，その生じるレベルにより，関数の引数・結果の関係において生じる外部変則性，および，関数本体内の計算において生じる内部変則性，の二つのレベルに区別される．このうち，未定義/未参照オブジェクトの検出は関数名・変数名のクロスリファレンス（cross-reference）により容易になされる．そこで以下では，不要/発散オブジェクトの検出についてのみ考察する．

変則性検出の対象は，外部変則性として，不要な関数引数および発散する関数定義，内部変則性として，不要な局所変数および発散する局所変数定義である．検出アルゴリズムは，図2に示すように計算経路解析のレベルに対応して，関数間・関数内の二つのレベルで行う．すなわち，関数間計算経路解析である PDPS 解析を用いて，外部変則性の検出を行う．更に，PDPS 解析結果に基づく関数内計算経路解析である VDPS 解析を用いて，内部変則性の検出を行う．

（1） 外部変則性と PDPS 解析との関係

PDPS の各要素である MSPS は，関数の計算経路上参照される引数の組である．そこで，外部変則性検出は，PDPS 解析結果を用い，以下のようにして行う．

まず，関数が不要な引数をもつかどうかは，関数のいかなる MSPS でも参照されない引数があるか否かで判定する．例えば，関数 $f(x, y, z)$ の PDPS を $\{\{x\}, \{x, y\}\}$ とすると，引数 z はいずれの MSPS にも含まれないので，不要引数であることがわかる．

関数が常に発散するかどうかは，PDPS が空集合であるか否かで判定する．なぜなら，MSPS は，関数の停止する計算経路と対応しているからである（定義関数の PDPS は，{ } ではなく，{{ }}，すなわち，1個の空集合である MSPS から成り立っていることに注意

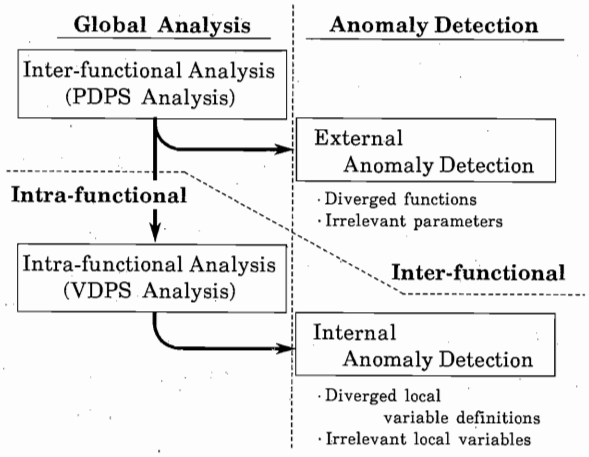

図2　2レベルの変則性検出と広域解析との関係
Fig. 2　Relation between 2-levelled anomaly detections and global dataflow analyses.

されたい).

(2) 内部変則性とVDPS解析の関係

VDPSの各要素であるMSVSは，関数の計算経路上参照される引数・局所変数・ラベル変数の組である．そこで，不要な局所変数および発散する局所変数定義などの内部変則性は，MSPSをMSVSと読み替えることにより，外部変則性検出の場合と同様にして行える．但し，VDPS解析はPDPS解析とは異なり，無限ループとなり本来発散する計算経路も解析結果に含めてしまう．そこで，計算結果を与える停止する計算経路のみを検出するためには，新たにループ検出が必要となる．

基本的には，無限ループは，変数のVDPSに自分自身が現れるか否かで判定する．例えば，変数aのVDPSが$\{\{b, c\}, \{a, b, c\}\}$である場合，後者のMSVSは変数$a$を含んでいる．これは，変数$a$の値を計算するのに変数$a$の値自身が必要であるという循環定義を意味し，無限ループであることがわかる．

遅延評価をもつ関数型言語のループ検出では，ストリームと無限ループを区別するため，変数参照モードが重要な役割を果たす．例えば，

{letrec　a=cons(1, a)；
　　b=cons-stream(1, b)；
　　c=car(a)；…}

とすると，$\mathcal{V}(a)=\{\{a\}\}$，$\mathcal{V}(b)=\{\{\bar{b}\}\}$，$\mathcal{V}(c)=\{\{a\}\}$となる．$a$，$b$の計算経路はともにループになっているが，値$a$の定義は循環定義となり発散してしまうのに比べ，値$b$は(1　1　1　…)というストリームを与える．この両者は，値aのMSVSが自らをSモードで参照し，値bのMSVSは自らをDモードで参照していることから判別できる．すなわち，自らをSモードで参照している場合(以下，Sモードループと呼ぶ)は，循環定義となり発散するのに対し，Dモードで参照している場合は，ストリームを生成する．

ある変数の発散は，それを参照する変数の発散性にも影響を与える．例えば，上の例における値cのMSVSのように、常に発散する値aをSモードで参照するMSVSも発散する．そこで，VDPS解析結果により与えられるデータの依存関係に基づき，発散性を伝搬させることにより，停止する計算経路のみを判別している．

3.2　外部変則性検出の例

前節で述べたように，不要な関数引数および発散する関数定義は，次のようにして判定する．

- x_iが関数$f(x_1, \cdots, x_n)$の不要引数
  $\Longleftrightarrow x_i, \bar{x}_i \notin$MSPS　for　$^{\forall}$MSPS
  $\in \mathcal{D}(f(x_1, \cdots, x_n))$
- $f(x_1, \cdots, x_n)$は発散する関数
  $\Longleftrightarrow \mathcal{D}(f(x_1, \cdots, x_n)) = \phi$

これは，MSPSは関数の停止する計算経路において参照する引数の組であるので，

- どのMSPSにも引数$x_i, \bar{x}_i$が含まれない．
  ⟺どの計算可能な計算経路においてもx_iをSモード・Dモードを問わず参照しない．
- MSPSが存在しない．
  ⟺計算可能な計算経路が存在しない．

と解釈できるからである.

以下において，それぞれの検出例を挙げる.

(例) easy(x, y)=if x==0 then 1 else
easy$(x-1,$ easy$(y, x))$

PDPS 解析より，$\mathcal{D}$(easy(x, y))$=\{\{x\}\}$ となるので，y は不要引数であることがわかる.

(例) diverged(x, y)=if $x+y<0$
then diverged$(x+1, y)$
else diverged$(x-1, y)$

PDPS 解析より，$\mathcal{D}$(diverged(x, y))$=\phi$ となるので，diverged(x, y) は発散することがわかる.

3.3 内部変則性検出の例

関数内レベルの変則性検出はソースプログラムにラベル付け変換を施したプログラムの VDPS 解析の解析結果に基づいて行われる.

[プログラム例]

$foo(x, y, z)$
=$\{$letrec $a=x+y$; $b=$cons(y, b) ;
$c=$cons-stream(y, c) ;
$d=$if $a>0$ then car(b) else
car(c) ;
in easy$(a,$ diverged$(y, z)*d)\}$

(但し，diverged(x, y)，easy(x, y) は **2.3** で定義されたとおりとする.)

このプログラム例においては，PDPS 解析結果，ラベル付け変換結果，VDPS 解析結果は，次のようになる.

[PDPS 解析] $\mathcal{D}(foo(x, y, z))=\{\{x, y\}\}$ (z は不要)

[ラベル付け変換結果]

foo(x, y, z)
=$\{$letrec $a=x+y$; $b=$cons(y, b) ;
$c=$cons-stream(y, c) ; $d1=a>0$;
$d2=$car(b) ; $d3=$car(c) ;
$d=$if $d1$ then $d2$ else $d3$;
$e=$diverged(y, z) ; $f=d*e$;
result$=$easy(a, f) ;
in result$\}$

[VDPS 解析]

$\mathcal{V}$(result)$=\{\{x, y, a\}\}$, $\mathcal{V}(a)=\{\{x, y\}\}$,
$\mathcal{V}(b)=\{\{y, b\}\}$, $\mathcal{V}(c)=\{\{z, \bar{c}\}\}$, $\mathcal{V}(e)=\phi$,
$\mathcal{V}(d)=\{\{x, y, a, b, d1, d2\},$
$\{x, y, a, c, \bar{c}, d1, d3\}\}$,
$\mathcal{V}(f)=\{\{x, y, a, b, d, d1, d2\},$
$\{x, y, a, c, \bar{c}, d, d1, d3\}\}$,
etc.

3.1 に述べたように，不要な局所変数は，次のようにして検出する.

・局所変数・ラベル変数 a が不要な変数
$\Longleftrightarrow a, \bar{a} \notin$MSVS for $^{\forall}$MSVS$\in\mathcal{V}$(result)

(i. e. どの return 式の計算経路においても a をSモード・Dモードを問わず参照しない.)

例えば，$foo(x, y, z)$ においては $\{z, b, c, d, d1, d2, d3, e, f\}$ は result の VDPS に現れないので不要な局所変数と判定される.

発散する局所変数定義の検出アルゴリズムは図3に示すように，循環定義となるSモードループを検出する Step 1，および，検出された発散性を他の局所変数定義およびその計算経路に対し波及させる Step 2～4 から構成される.

[Step 1] 循環定義となるSモードループの検出 (Initial Path Anomaly Detection)

発散する局所変数定義の初期値として，Sモードループとなる計算経路 (MSVS) をもつ局所変数を用いる.

例えば，関数 foo の例において，図4の太線部分に対応する $b=$cons(y, b) の MSVS$\{y, b\}\in\mathcal{V}(b)$ は $b\in\{y, b\}$ であるので，Sモードループである.

[Step 2] 計算経路から局所変数定義へ発散性の波及

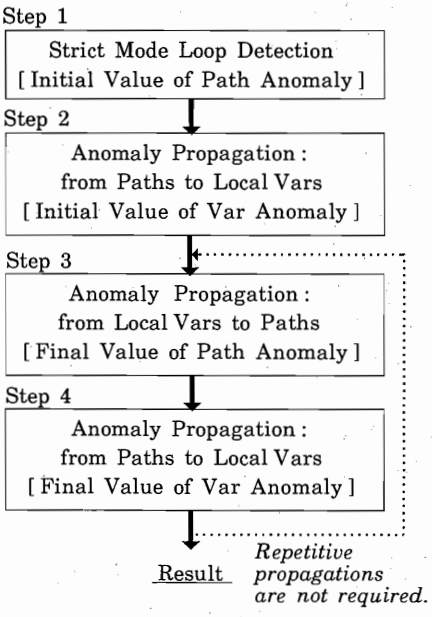

図3 内部発散性検出アルゴリズム
Fig. 3 Internal anomaly detection algorithm for diverged objects.

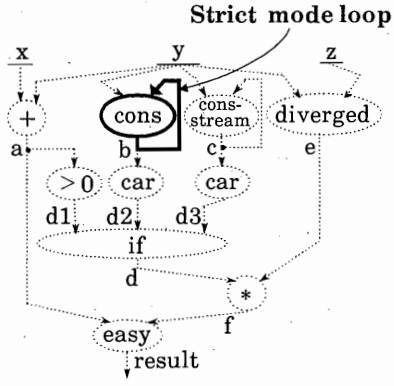

図4　Sモードループの検出（ステップ1）
Fig. 4　Strict model loop detection (Step 1).

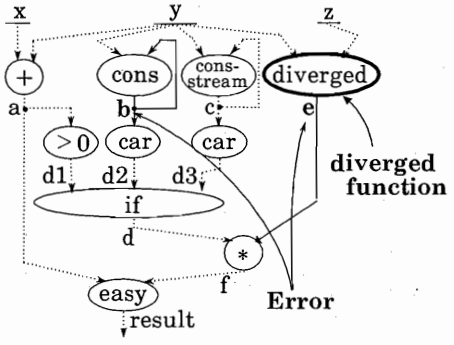

図5　局所変数への発散性の伝搬（ステップ2）
Fig. 5　Anomaly propagation to local variables (Step 2).

（Anomaly Propagation：Initial Var Anomaly）

Step 2はStep 1で求めたSモードループにより生じるMSVSの発散性を，そのMSVSが属する局所変数定義へ波及させる．波及の規則は表1に示すとおりである．例えば，図5において太字で表される局所変数 b の局所定義 $b=\mathrm{cons}(y, b)$ は，エラーとなるMSVS$\{y, b\}$ しかもたないので，表1の「すべてのMSVSがエラー」に対応し，エラーと判定される．更に，Step 2においては，局所変数定義内における発散する関数の適用により生ずる発散性も検出する．例えば，図5において太字で表される局所変数 e は，変数 y，z に発散する関数 divergedを適用して定義され，$\mathcal{V}(e)=\phi$ となるので，表1の「VDPSが空集合」の場合に対応して，エラーと判定される．

[Step 3]　局所変数から計算経路へ未定義性の波及

（Anomaly Propagation：Final Path Anomaly）

Step 3はStep 2で検出された局所変数のエラーワーニングをその局所変数をSモードで参照している計算経路（MSVS）へ波及させる．波及の規則は表2に示すとおりである．例えば，MSVS$\{x, y, a, b, d1, d2\}\in\mathcal{V}(d)$ は，エラーとなる局所変数 b をSモードで含むので，表2の「MSVS内Sモード変数に含まれるもっとも重大なエラー」が「エラー」に対応し，エラーと判定される．このStep 3で求まる，エラーとなるMSVSを除いたVDPSが，発散する計算経路を省いたVDPSとなる．

表1　計算経路から局所変数へのエラー（発散性）の波及

局所変数のVDPSに含まれるMSVSのエラー程度	すべてのMSVSがエラー	VDPSが空集合	一部のMSVSがエラーまたはワーニング
導かれる局所変数のエラー程度	エラー	エラー	ワーニング

表2　変数から計算経路へのエラー（発散性）の波及

MSVS内Sモード変数に含まれる最も重大なエラー	エラー	ワーニング
導かれるMSVSのエラー程度	エラー	ワーニング

[Step 4]　計算経路から局所変数定義へ未定義性の波及

（Anomaly Propagation：Final Var Anomaly）

Step 4はStep 3で検出された計算経路のエラーワーニングをその計算経路の属する局所変数定義へ波及させる．発散性の波及の規則はStep 2と同様である．図6において太線で表される部分は，エラーの原因となる計算経路，もしくはエラーを伝搬する計算経路であり，太点線で表される部分は，ワーニングを伝搬する計算経路である．図6において太字で表される局所変数 d は，$\mathcal{V}(d)$ がエラーとなるMSVS$\{x, y, a, b, d1, d2\}$ およびそうでないMSVS$\{x, y, a, c, \bar{c}, d1, d3\}$ の双方をもつので，ワーニングと判定される．局所変数 f のVDPSは，$\{\{x, y, a, b, d, d1, d2\}, \{x, y, a, c, \bar{c}, d, d1, d3\}\}$ であり，いずれのMSVSもエラーであるので，エラーと判定される．しかし，出力resultにおいては，$\mathcal{V}$(result)のMSVSはいずれも f を含まず，easy(a, f) の f が不要であるので，これらの発散性は波及しない．

この発散性伝搬アルゴリズムにおいて，VDPS解析の性質により，各計算経路はデータの依存関係に基づく推移的閉包に達している．そのため，図3の点線で

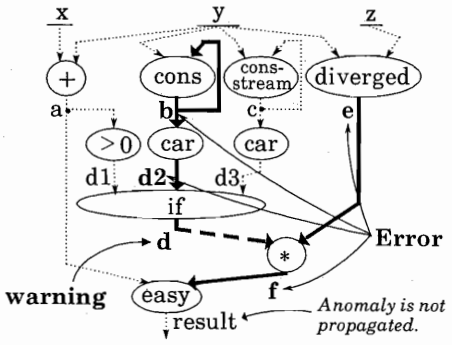

図6 局所変数への発散性の伝搬(ステップ4)
Fig. 6 Anomaly propagation to local variables (Step 4).

表されたようなStep 2～4の波及アルゴリズムを更に繰り返しても，新たな変数や計算経路に発散性が波及することはない．

なお，より大きな実例，例えばDijkstraのhamming数(1および2，3，5の倍数を小さい方から並べたもの)については，文献(16)を参考にされたい．

4. 変則性検出アルゴリズムの検出能力

4.1 他の解析手法との比較

本節では，従来より用いられている構文上の解析であるクロスリファレンスの手法，および，ストリクトネス解析に基づく手法の限界を，不要/発散オブジェクト検出の面から例を挙げて明らかにする．

クロスリファレンスは未定義/未参照オブジェクトの検出に用いられる．例えば，

$\text{my-if}(x, y) = \text{if}(x, y, z)$,

$\text{my-if}'(x, y, z, u) = \text{if}(x, y, z)$

において，それぞれ左辺に現れる変数の集合と右辺に現れる変数の集合を比較する．$\text{my-if}(x, y)$の本体内のzは，右辺に現れるが左辺に現れない未定義変数であり，$\text{my-if}'(x, y, z, u)$の$u$は左辺に現れるが右辺に現れない未参照変数と検出される．

しかし，不要オブジェクトの検出に応用しようとすると，構文上存在するものを，その意味に立ち入って計算結果に与える影響を理解することができないので，

$$\text{easy}(x, y) = \text{if}\quad x = 0\quad \text{then}\quad 1\quad \text{else}\quad \text{easy}(x-1, \text{easy}(y, x))$$

のように，左辺・右辺の双方に存在していても，実は，第2引数yの評価が計算結果に影響を与えないため，yが不要となることを検出できない．

ストリクトネス解析は，広域解析の一つとして，従来広く用いられてきた解析手法であり，関数を評価する上で必要となる引数を解析する．しかし，発散オブジェクトの検出に応用しようとすると，発散する関数はストリクト関数とみなして解析するため[8],[9]，検出できない．例えば，

$$\text{diverged}(x, y) = \text{if}\quad x + y < 0\quad \text{then diverged}(x+1, y)\quad \text{else diverged}(x-1, y)$$

$$\text{strict}(x, y) = \text{if}\quad x + y < 0\ \text{then}\quad x + y\quad \text{else}\quad \text{strict}(x-1, y)$$

において，$\text{diverged}(x, y)$は発散する関数であり，$\text{strict}(x, y)$はストリクトな関数である．これを計算経路解析では，$\mathcal{D}(\text{diverged}(x, y)) = \phi$，$\mathcal{D}(\text{strict}(x, y)) = \{\{x, y\}\}$として，判別するが，ストリクトネス解析ではともに，$\{x, y\}$となり，判別することができない．

4.2 変則性検出アルゴリズムの停止性

変則性検出のアルゴリズムの停止性は，計算経路解析アルゴリズムの停止性と，関数内発散オブジェクト検出における発散性伝搬アルゴリズムの停止性により保証される．

データ構造のない領域上の関数を対象とする計算経路解析の停止性は，既に示されている[13]．データ構造のある領域上の関数への拡張として，Sモード・Dモードのように有限個のモードを付与した計算経路解析も停止性を満たすことは明らかである．

また，各部分式の計算経路(MSVS)は，その部分式の依存する変数の集合のデータの依存関係に基づく推移的閉包であるので，発散性の伝搬は各計算経路ごとに一度伝搬させれば十分である．従って発散性伝搬アルゴリズムの停止性も保証されている．

このようなコンパイル時の解析手法においては，停止性と解析力はトレードオフの関係にある．例えば，より精密な計算経路解析として，各引数部に対する評価要求がその引数が示すデータ構造内のどの部分構造に伝搬するかを，正確に解析することも考えられる．しかし，このような場合，無限個の部分構造をもつデータ構造を考えれば明らかなように，停止性が保証されない．本論文で採用している方法は，データ構造の内部構造には立ち入らず，データ構造を生成するときに，どの値を評価して参照し，どの値を遅延して参照しているか，という点のみに着目している．このため，再帰的なデータ構造に対しても停止性が保証されている．

4.3 変則性検出アルゴリズムの安全性

前節で述べたように，コンパイル時の解析手法は，停止性を保証するため解析力を制限しなければならな

い．従って，アルゴリズムにより求まるプログラムの性質は，実行時の挙動により定まる真の性質の近似となる．このとき問題となるのは，近似の妥当性である．これを安全性（safeness）と呼ぶ．

計算経路解析に基づく変則性検出における安全性は，エラーでないオブジェクトはエラーと判定しないということになる．これは，

・不要でないオブジェクトを不要と判定しない．

・発散しないオブジェクトを発散すると判定しない．

を意味する．

一方，計算経路解析は，

・PDPS 解析結果は真の PDPS を含む．

・VDPS 解析結果は真の VDPS を含む．

という性質を満たしている[13),(14)]．これは，

・解析により求まる計算経路は真の計算経路を含む．

ということを意味し，**3.1** の検出法より計算経路解析に基づく変則性検出は安全性を満たすことを示すことができる．

また，本アルゴリズムにおいては，関数内発散オブジェクトの検出において，エラーとは限らないが，エラーとなる可能性がある（エラーとなる計算経路をもつ）オブジェクトを，エラーとは別にワーニングとして検出し，安全性を保ちつつ解析力を補っている．

4.4 変則性検出アルゴリズムの限界

本アルゴリズムの限界には次の 2 種類がある．

（1） 本アルゴリズムは，すべての計算経路が無限再帰または無限ループになる場合にのみ，その式を発散すると判定する．従って，

$$f\text{-error}(x)=\text{if}\quad x^2<0\quad\text{then}\quad 1\quad\text{else}\quad f\text{-error}(x+1)$$

のように，実行すると常に発散してエラーになるが，みかけ上計算可能な計算経路（if 式の判定部が true となる場合）がある関数は検出されない．また，

$$f\text{-normal}(x)=\text{if}\quad x^2>0\quad\text{then}\quad 1\quad\text{else}\quad \text{diverged}(x,1)$$

（但し，$\text{diverged}(x,y)$ は $\mathcal{D}(\text{diverged}(x,y))=\phi$ となる，常に発散する関数である）のように実行すると常に 1 を返す関数でも，みかけ上発散する経路（if 式の判定部が false となる場合）がある関数には，関数内解析時にワーニングを発する．但し，いずれの場合も，エラーでないものをエラーと判定することはないので，解析の安全性は保たれている．

（2） 本論文で用いた変則性検出アルゴリズムは，MSPS･MSVS に含まれる D モードの引数に対し，その引数部を評価した結果，生じるエラーは解析していない．従って実行時にその引数部を評価し，それがエラーになる場合，例えば

$$f\text{-str-error}(x)=\{\text{letrec}\quad a=\text{cons-stream}(x,\text{cdr}(a))\quad\text{in}\quad \text{car}(\text{cdr}(a))\}$$

などの関数の変則性（発散性）は検出できない．

より解析の精度を上げても，すべての発散オブジェクトの検出はプログラムの停止性の判定と等価になり，不可能である．このように，静的な解析・検出においては必ず検出不能な変則性が存在する．

5. むすび

本論文では一階の関数型プログラムを対象とし，計算経路解析を用いた新たな変則性検出アルゴリズムを提案した．

変則性検出は，参照されるがその値が結果に影響を与えない不要オブジェクト，および，出口のない無限再帰に陥り停止しない発散オブジェクトの 2 種類について行った．また，解析は関数の引数・結果の関係において生じる外部変則性と，一つの関数本体内の計算において生じる内部変則性との 2 レベルで行った．

この検出アルゴリズムの基礎となる解析として，外部変則性・内部変則性に対し，それぞれ，引数依存属性集合解析･部分値依存属性集合解析を用いた．また，これらの解析においては，発散するループとなる計算経路と，ストリームを生成する計算経路とを区別するため，S モード・D モードの変数参照モードの概念を用いた．

本アルゴリズムはストリームのように，本来無限の計算を含むデータを処理するプログラムにおいて生じやすい，計算が発散するバグのコンパイル時自動検出に有効である．なお，本アルゴリズムの確認のため，Common-Lisp を用いて試作システムを作成した．規模は，広域解析部分が約 2 kstep，変則性検出部分が約 1 kstep である．今後は，試作システムを関数型プログラムの開発環境と統合し，より現実的なプログラムに対し，本アルゴリズムの有効性を確認していきたい．

謝辞 日ごろご指導くださる九州大学の雨宮真人教授，ならびに，本論文に関してご討論くださった NTT ソフトウェア研究所ソフトウェア基礎技術研究部の高橋直久主任研究員をはじめとするデータフロー研究グループ諸氏に感謝いたします．

文　献

(1) P. Henderson : "Functional Programming, Application and Implementation", Prentice-Hall (1980).

(2) A. V. Aho, R. Sethi and J. D. Ullman : "Compilers, Principles, Techniques and Tools", Addison Wesley (1986).

(3) H. Abelson, G. J. Sussman and J. Sussman : "Structure and Interpretation of Computer Programs", MIT Press (1985).

(4) L. D. Fosdick and L. J. Osterweil : "Data flow analysis in software reliability", ACM Computing Surveys, 8, 3, pp. 305-330 (1976).

(5) J. Jachner and V. K. Agarwal : "Data flow anomaly detection", IEEE Trans. Software Eng., **SE-10**, 4, pp. 432-437 (1984).

(6) A. Mycroft : "The theory and practice of transforming call-by-need into call-by-value", Lecture Notes in Comput. Sci., **83**, pp. 269-281 (1980).

(7) J. Fairbairn and S. C. Wray : "Code generation techniques for functional languages", Proc. 1986 ACM Conf. LISP and Functional Programming, pp. 94-104 (1986).

(8) C. Clack and S. L. P. Jones : "Strictness analysis — Practical approach", Lecture Notes in Comput. Sci., **201**, pp. 35-49 (1985).

(9) P. Hudak and J. Young : "Higher-order strictness analysis in untyped lambda calculus", Proc. 13th ACM sympo.on Principles of Programming Languages, pp. 97-109 (1986).

(10) S. Ono : "Comparison among strictness-related analyses for applicative languages", 信学技報, **COMP87**-38 (1987-10).

(11) S. Ono, M. Takahashi and M. Amamiya : "Non-Strict Partial Computation with a Dataflow Machine", 京都大学数理解析研究所講究録, **547**, pp. 196-229 (1985).

(12) S. Ono, N. Takahashi and M. Amamiya : "Optimized demanddriven evaluation of functional programs on a dataflow machine", IEEE Proc. 15th Inter. Conf. Parallel Processing, pp. 421-428 (1986).

(13) 小野，高橋："再帰関数系における依存属性集合の計算法"，信学論(D)，**J69-D**, 5, pp. 714-723 (昭 61-05).

(14) 小野，高橋："関数型言語における要求駆動型評価の並列処理向き最適化"，信学論(D)，**J70-D**, 2, pp. 259-268 (昭 62-02).

(15) M. Ogawa and S. Ono : "Transformation of strictness-related analyses based on abstract interpretation", Inter. Conf. Fifth Generation Computer Systems 1988 (1988).

(16) 小川，小野："広域データフロー解析に基づく関数型プログラムの変則性検出"，データフローワークショップ，信学会データフローアーキテクチャと並列処理時限研究専門委員会，pp. 33-40 (1987).

(昭和 63 年 3 月 10 日受付)

小川　瑞史

昭 58 東大・理・数学卒．昭 60 同大大学院修士課程了．同年 NTT 武蔵野電気通信研究所入所．以来，関数型言語の研究に従事．現在，NTT ソフトウェア研究所研究主任．情報処理学会，ACM 各会員．

小野　諭

昭 52 東大・工・電子卒．昭 57 同大大学院博士課程了．同年日本電信電話公社武蔵野電気通信研究所入所．以来，関数型言語および並列計算機の研究に従事．現在，NTT ソフトウェア研究所主任研究員．工博．情報処理学会，ACM 各会員．