

経路依存フローグラフを用いた プログラム・スライシング

直井 邦彰 高橋 直久

NTT ソフトウェア研究所
東京都港区港南 1-9-1 NTT 品川 TWINS

あらまし 本稿では、一般化したデータフロー計算モデルで先に提案した経路依存フローグラフ (PDFG) を並列実行させることにより、各種スライスを統一的なフレームワークで求める手法を提案する。本稿では、まず、静的 / 動的、逆方向 / 順方向、実行可 / クロージャと呼ぶ 3 つの属性により、一般化した形式でスライスを表現する。次に、スライス作成のために、各属性に対して PDFG のノードの実行規則を与える。更に、3 つの属性のとり値の組合せとして定義される 8 通りのスライスに対して、これらの実行規則を用いた計算法を与える。PDFG では、データ、経路、制御の依存関係がアークとして表現されているため、ノードの実行規則を入れ換えることにより、各種スライスの検出が可能となっている。また、本稿では、従来のスライスを含むよう一般化したため、提案した手法では、既存のスライスおよび、従来考慮されていなかったスライスを求めることが可能となった。

和文キーワード プログラム解析, データフロー型計算モデル, 依存グラフ, 手続き型プログラム, 並列実行, プログラム・スライス

Program Slicing Using a Path Dependence Flow Graph

Kuniaki NAOI Naohisa TAKAHASHI

NTT Software Laboratories
Kohnan, 1-9-1, Minato-ku, Tokyo 108, JAPAN

Abstract This paper presents a unified framework for detecting various kinds of program slices. Here, the path dependence flow graph (PDFG) presented earlier is interpreted in parallel based on a generalized dataflow computing model. This paper defines eight kinds of slices as combinations of three binary-value attributes, and three execution rules corresponding to these attributes. All slice calculations can be attained by executing PDFG nodes with execution rules corresponding to slice attributes. This is because the data, path and control dependencies of a program are represented as arcs in PDFG.

英文 key words program analysis, dataflow computation model, dependence graph, imperative program, parallel execution, program slice

1 まえがき

プログラムから、ある着目する性質を満たす文集合(スライスという)を抽出する技術はプログラム・スライシングと呼ばれ、プログラムの簡単化、デバッグや保守などのために広く用いられている¹⁾⁻¹¹⁾。これまでに、様々な種類のスライスが提案され、それぞれ作成法が与えられている¹⁾⁻⁹⁾。

例えば、データフロー等式の反復計算によるスライス作成法が提案されている。しかし、この方法は、スライスの作成手順が複雑であるとともに、異なるスライスを求める場合、それぞれ異なる作成手順を与える必要があるという問題がある。一方、依存グラフにおける到達可能性判定によりスライスを作成する方法が提案されている。この方法は、スライスを作成する手順が直観的で分かりやすいという特長がある。しかし、これまでに提案されてきたスライス作成法では、スライスの性質ごとに依存グラフの表現法を定め、それぞれ異なるグラフ操作法を与えている。例えば、静的スライスを求めるための依存グラフ⁴⁾では、実行時に生成される依存関係を十分に表現していないので、動的スライスを求めるために用いることができないという問題があった。

本稿では、一般化したデータフロー計算モデルで先に提案した経路依存フロウグラフ(PDFG)を並列実行させることにより、各種スライスを統一的なフレームワークで求める手法を提案する。本稿では、まず、静的/動的、逆方向/順方向、実行可/クロージャと呼ぶ3つの属性により、一般化した形式でスライスを表現する。次に、スライス作成のために、各属性に対してPDFGのノードの実行規則を与える。更に、3つの属性のとり値の組合せとして定義される8通りのスライスに対して、これらの実行規則を用いた計算法を与える。

提案手法では、スライスの作成手順はノードの実行規則として局所的に与えればよいので、スライス作成の計算制御が簡明である。更に、PDFGでは、データ、計算経路、制御の依存関係がアークとして表現されており、ノードの実行規則の入れ換えにより、各種スライスの作成を可能としている。

本稿の構成は次の通りである。まず2では、プログラム・スライシングの概念を一般化する。3では、PDFGの定義を示す。4では、PDFGのアークを辿ることにより得られる各種の依存関係を定める。5では、一般データフロー計算(GDC)と、その実行機構について述べる。6では、GDCモデルに基づくPDFGの実行法について述べる。7では、PDFGのノードの実行規則を示し、GDCモデルに基づく実行規則の解釈実行によるスライス作成法を提案する。8では、スライスの作成例を示す。

2 プログラム・スライシング

ソースコードから、注目する性質を持つ文集合を抽出する技術をプログラム・スライシングと呼び、抽出したコードをスライスと呼ぶ¹⁾。求めたい性質に応じて様々なスライシング技法¹⁾⁻⁸⁾が提案されており、これらは表1に示す通り分類できる¹¹⁾。

表1 スライシング技法の分類

入力 (f_i)	方向 (f_d)	実行可能性 (f_e)	
(全入力) 静的 (st)	逆方向 (bk)	実行可(ex)	文献(1), 文献(2), 文献(3)
		クロージャ(cl)	文献(4)
	順方向 (fw)	実行可(ex)	—
		クロージャ(cl)	文献(4)
(単一入力) 動的 (dy)	逆方向 (bk)	実行可(ex)	文献(5)
		クロージャ(cl)	文献(6), 文献(7), 文献(8)
	順方向 (fw)	実行可(ex)	—
		クロージャ(cl)	—

すなわち、これまで提案されてきたスライスは、入力 f_i 、方向 f_d 、実行可能性 f_e と呼ぶ3つの属性¹⁰⁾に従い分類される。

各属性の性質は次の通りである。逆方向($f_d=bk$)のスライスは、プログラムの実行方向とは逆の方向に、着目する文のある変数から、変数間の依存関係を辿ることによって到達できるすべての文を含む。順方向($f_d=fw$)のスライスは、実行と同じ方向に辿ることによって到達できるすべての文を含む。そして、実行可($f_e=ex$)のスライスは実行可能であり、スライスを実行させた際、着目する文に対して元のプログラムと同じ振舞いをする。一方、クロージャ($f_e=cl$)のスライスは、必ずしも実行可とは限らない。更に、静的($f_i=st$)は、以上で述べた各性質が、すべての入力に対して成り立つことを意味する。一方、動的($f_i=dy$)は、ある特定の入力に対してのみ成り立つことを意味する⁹⁾。

ここで、(f_i, f_d, f_e)をスライス属性と呼び、スライスの性質を表すために用いる。例えば、(st, bk, ex)のスライスは、静的逆方向実行可のスライスを表す。(st, bk, ex)のスライスはプログラムの簡単化や並列化などに、(st, fw, cl)のスライスは影響波及解析などに、(dy, bk, ex)のスライスはバグ検出などに用いられる。

また、スライスを求める対象となるプログラムを P 、 P への入力値を I 、着目する文を S 、着目する変数の集合を V としたとき、これらの値とスライス属性の7つ組($P, I, S, V, f_i, f_d, f_e$)によりスライスを表す。ここで、 I は $f_i=dy$ のときに意味を持つ。また、 $f_i=dy$ のとき、文献(9)のように $S=s^q$ と表した場合には、 P の実行履歴において q 番目に現れる文 s から辿り始めることを意味する。

3 経路依存フロウグラフ

我々は、先に、依存関係に基づくプログラム解析を統一的な枠組の上で行える、経路依存フロウグラフ(Path Dependence Flow Graph: PDFG)¹²⁾⁻¹⁴⁾と呼ぶ有向グラフ表現を提案した^{15), 16)}。

PDFGは、手続き型プログラムに対する表現形式であり、プログラムにおける、データ、計算経路、制御の3つの依存関係を表現している。更に、プログラムの実行に必要なすべての情報を保持しており、データフロー計算モデル¹⁷⁾に基づき並列実行が可能である。データフロー計算モデルでは、演算結果のデータあるいは実行順序を示す信号はトークンと呼ばれ、トークンがグラフ上を流れることにより計算が進められる。

3.1 経路依存フロウグラフの表記

ノード集合 N_0 、有向アーク集合 A_0 、開始ノード n_s 、終了ノード n_e を用いて、PDFGを $G=(N_0, A_0, n_s, n_e)$ と表す。

ノード $n(n \in N_0)$ は任意個の入力ポートと出力ポートを持ち、 n の入力ポート集合を $I(n)$ 、出力ポート集合を $O(n)$ と表す。そして、2つのノード n_i, n_j に対して、 n_i の出力ポート x から n_j の入力ポート y への有向枝をアーク、 x をアークの始点、 y をアークの終点と呼び、 $x \rightarrow y$ と表す。そして、入力アークを持たないノードを開始ノード、出力アークを持たないノードを終了ノードと呼ぶ。更に、ポート x が属するノードを $node(x)$ と表す。

3.2 ノード集合

PDFGのノード集合 N_0 の要素の属性間に半順序関係を設定し、最大元 $\top$ と最小元 $\perp$ を加えてできる半順序集合を N_0 のノード属性集合といい、 $N(N_0)$ と表す。以下、順序関係を $\prec$ により表し、 $a \prec b$ のとき b は a より大きいという。また、 $N(N_0)$ 上の半順序関係を図1に示す。ここで、 $start$, end , $switch$ -macro, $merge$ -macro, $operational$ の各属性は、デ

ータフロー計算モデルにおける開始, 終了, *switch*, *merge*, 演算の各ノード種別に対応する。そして, *load*, *store* の両属性は, グローバルメモリを読み書きするノードを示す。

ただし, *switch-macro* 属性のノードは, 同じ文に対応する複数の *switch* ノードを一つのノードで表現したものであり, 以下の通り定義する。

[定義 1] *switch-macro* ノード。

switch-macro 属性のノードは, 2 組の入力 $\langle x_1, x_2, \dots, x_m \rangle$, y と, 2 組の出力 $\langle z_{t_1}, z_{t_2}, \dots, z_{t_m} \rangle$, $\langle z_{f_1}, z_{f_2}, \dots, z_{f_m} \rangle$ を持つ分岐命令であり, y から入力したトークンの値が真ならば z_{t_i} に, 偽ならば z_{f_i} に, x_i ($1 \leq i \leq m$) からの入力トークンを出力する。ここで, x_i を分岐対象入力ポート, y を真偽決定入力ポート, z_{t_i} を真方向出力ポート, z_{f_i} を偽方向出力ポートとそれぞれ呼ぶ。

同様に, *merge-macro* 属性のノードは, 同じ文に対応する複数の *merge* ノードを一つのノードで表現したものであり, 以下の通り定義する。

[定義 2] *merge-macro* ノード。

merge-macro 属性のノードは, 2 組の入力 $\langle x_1, x_2, \dots, x_m \rangle$, $\langle y_1, y_2, \dots, y_m \rangle$ と, 1 組の出力 $\langle z_1, z_2, \dots, z_m \rangle$ を持つ *merge* 命令であり, x_i および y_i ($1 \leq i \leq m$) からトークンを入力したとき, それぞれ z_i にトークンを出力する。

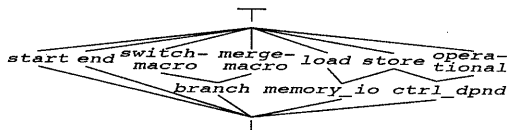


図 1 経路依存フローグラフのノード属性集合
Fig. 1 A set of node attributes on a PDFG.

以下では, ノード属性集合のある属性を指定することは, その属性より大きいか等しい属性をすべて指定したことを意味する。例えば, *memory.io* ノードという場合には, *load* ノードと *store* ノードを意味する。

3.3 アーク集合

3.2 と同様に, PDFG のアーク集合 A_0 の要素の属性間に半順序関係を設定し, $\top$ と $\perp$ を加えてできる半順序集合を A_0 のアーク属性集合といい, $A(A_0)$ と表す。そして, $A(A_0)$ 上の半順序関係を図 2 に示す。ここで, *path* は *branch* ノード間の順序関係を, *imperative* は *memory.io* ノード間の順序関係を, そして *functional* は, *memory.io* および *operational* ノード相互間でのデータ授受関係を, それぞれ示す。

また, *store_trm* は, *imperative* $\prec$ *store_trm* を満たす, 終点が *store* ノードであるアーク, *non_store_trm* は, *imperative* $\prec$ *non_store_trm* を満たす, 終点が *store* ノードではないアークを示す。

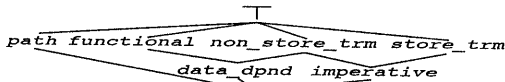


図 2 経路依存フローグラフのアーク属性集合
Fig. 2 A set of arc attributes on a PDFG.

ノード属性集合の場合と同様に, アーク属性集合のある属性を指定することは, その属性より大きいか等しい属性をすべて指定したことを意味する。

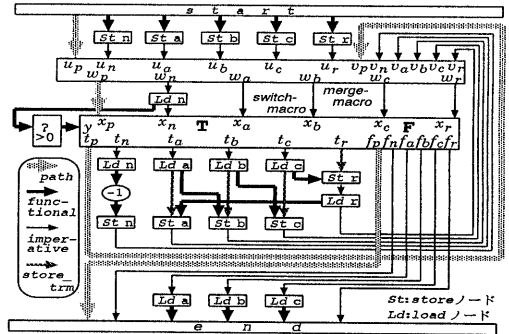
3.4 記述例

図 3(a) に示すプログラムの PDFG の例を図 3(b) に示す。ここで, 図 3(a) は, 3 変数 (x, y, z) の値を u 回右にローテート

させる計算を行うプログラムである。関数の引数のうち $*p_a$, $*p_b$, $*p_c$ は変数 x, y, z に, v_n は回数 u に対応する。

```
void rotation(p_a, p_b, p_c, v_n)
int *p_a, *p_b, *p_c, v_n;
{
    int a, b, c, n, r;
    a=*p_a; b=*p_b; c=*p_c; n=v_n;
    while(n-- > 0){
        r=c; c=b; b=a; a=r;
    }
    *p_a=a; *p_b=b; *p_c=c;
}
```

(a) ローテートを計算するプログラム
(a) A program for rotation.



(b) (a) のプログラムの PDFG の例
(b) An example of a PDFG corresponding to program (a).

図 3 サンプルプログラム (その 1)

Fig. 3 A sample program (1).

3.5 並列実行法

記述言語の文法に従う演算規則を PDFG のノードに付与し, データフロー計算モデルに基づいて各ノードを発火させると, PDFG の並列実行が行える。このとき, *memory.io* ノードでの機能に応じて, 次の 2 種類の実行が可能である。

(1) メモリアクセス型並列実行

memory.io ノードにおいて, 実際にメモリの読み書きを行いながら計算を進める。

(2) データ授受型並列実行

メモリの読み書きは行わず, *imperative* アークにトークンを送ることにより計算を進める。

以下では, PDFG の並列実行と呼ぶ場合, データ授受型並列実行を指すものとする。

4 PDFG における依存関係

スライスの計算法は, データ依存の関係と制御依存の関係を順次辿る操作として定められる⁸⁾。ここでは, 3 で定義した PDFG を用いてスライスを求める準備として, PDFG のグラフ構造の上でデータ依存関係と制御依存関係とを与え, 更に, 依存ノード集合を与える。

4.1 ポート間関係の定義

PDFG G の任意の 2 つのポートに対し, それぞれの性質に応じて依存関係を以下の通り定義する。

[定義 3] ノード間依存関係。

G のアーク $x \rightarrow y$ が属性 u を持つとき, y は x に u のノード間依存であると呼び, $x \xrightarrow{u} y$ と表す。

[定義 4] ノード内データ依存関係。

G のノード n において, $x \in I(n), y \in O(n)$ なる任意の x, y に対して, y から送信するトークンの値が x から受信したトークンの値に依存する場合, x から y へはノード内データ依存属性 D のポート間関係があるといひ, $x \xrightarrow{D} y$ と表す。また, 簡単に, y は x にノード内データ依存であるという。

[定義 5] ノード内制御依存関係。

$\mathcal{G}$ のノード n において, $x \in \mathcal{I}(n)$, $y \in \mathcal{O}(n)$ なる任意の x, y に対して, y からトークンを送信するかどうかは x から受信したトークンの値に依存する場合, x から y へはノード内制御依存属性 C のポート間関係があるといい, $x \xrightarrow{C} y$ と表す。また, 簡単に, y は x にノード内制御依存であるという。□

PDFG のノード定義から, 以下の通り依存関係が存在する。

[性質 1] *switch-macro* ノード内の依存関係。

n を *switch-macro* ノード, m を n の分岐対象入力ポートのポート数, x_i, y, z_i, z_f ($1 \leq i \leq m$) をそれぞれ n の分岐対象入力ポート, 真偽判定入力ポート, 真方向出力ポート, 偽方向出力ポートとすると, $1 \leq i \leq m$ に対して, $x_i \xrightarrow{D} z_i, x_i \xrightarrow{D} z_f$, および, $y \xrightarrow{C} z_i, y \xrightarrow{C} z_f$ が成立する。□

x_i からのトークンを z_i に送信するか z_f に送信するかが, y の値によって決まることに対応する。

[性質 2] *merge-macro* ノード内の依存関係。

n を *merge-macro* ノード, m を出力ポート数, x_i, y_i ($1 \leq i \leq m$) を入力ポート, z_i ($1 \leq i \leq m$) を出力ポートとすると, $1 \leq i \leq m$ に対して, $x_i \xrightarrow{D} z_i, y_i \xrightarrow{D} z_i$ が成立する。□

x_i および y_i から受信したトークンをそれぞれ, z_i に送信することに対応する。

[性質 3] *memory-io* ノード内の依存関係。

n を *memory-io* ノード, x, y を n の *data_dpnd* アークの終点と始点とすると, x からトークンを受信した場合 y にトークンを送信するため, $x \xrightarrow{D} y$ が成立する。□

[性質 4] *operational* ノード内の依存関係。

n を *operational* ノード, x を n の任意の入力ポート, y を n の出力ポートとすると, y へ送信するトークンの値の計算に x から受信するトークンの値が用いられるため, $x \xrightarrow{D} y$ が成立する。□

4.2 ポート間関係に対する演算

PDFG $\mathcal{G}$ 上で定義されるポート間関係における属性に対して, 次の演算を定義する。

[定義 6] ポート間関係の演算。

$$\begin{aligned} x \overset{u}{\rightarrow} y &\triangleq (x \overset{u}{\rightarrow} z) \wedge (z \overset{u}{\rightarrow} y) \\ x \overset{u}{\vee} y &\triangleq (x \overset{u}{\rightarrow} y) \vee (x \overset{u}{\rightarrow} y) \\ x \overset{u}{\rightarrow} y &\triangleq x \overset{u}{\rightarrow} (u \overset{u}{\rightarrow} y) \end{aligned}$$

ここで, x, y, z は $\mathcal{G}$ の任意のポート, u, v は任意の属性である。□

4.3 データ依存関係の定義

PDFG $\mathcal{G}$ において, 以下の通りデータ依存関係を定義する。

[定義 7] データ依存関係。

n_i, n_j を $\mathcal{G}$ の任意のノードとする。ここで, $x \in \mathcal{I}(n_i), y \in \mathcal{I}(n_j)$ なる任意のポート x, y に対して $x \xrightarrow{(D \text{ data_dpnd})^*} y$ が成立するとき, y は x にデータ依存であるといい, $x \xrightarrow{Data} y$ と表す。□

データ依存関係は, プログラム P において, 文 i で参照される変数 v_i の値が, 文 j で代入される変数 u_j の値に応じて変わること意味する。

4.4 制御依存関係の定義

[定義 8] 到達依存関係。

PDFG $\mathcal{G}$ において, n_i を *switch-macro* ノード, $\mathcal{O}^T(n_i)$, $\mathcal{O}^F(n_i)$ をそれぞれ n_i の真方向出力ポート, 偽方向出力ポートの集合とする。また, $\mathcal{G}$ のノード集合に対して, 真方向到達可能ノード集合 $\mathcal{N}_r^T(n_i)$, 偽方向到達可能ノード集合 $\mathcal{N}_r^F(n_i)$, 到達依存ノード集合 $\mathcal{N}_r(n_i)$ と呼ぶ 3 つの部分集合を, 次の通り定める。

- $\mathcal{N}_r^T(n_i) = \{node(y) | z \in \mathcal{O}^T(n_i), z \xrightarrow{Data} y\}$
- $\mathcal{N}_r^F(n_i) = \{node(y) | z \in \mathcal{O}^F(n_i), z \xrightarrow{Data} y\}$
- $\mathcal{N}_r(n_i) = \{n_i | n_i \in (\mathcal{N}_r^T(n_i) \cap \mathcal{N}_r^F(n_i)) \cup (\mathcal{N}_r^T(n_i) \cap \mathcal{N}_r^F(n_i))\}$

このとき, $n_j \in \mathcal{N}_r(n_i)$ は n_i に到達依存であるといい, $n_i \xrightarrow{Rdpnd} n_j$ と表す。□

到達依存関係は, PDFG において, *switch-macro* ノード n_i から *data_dpnd* アークを出力アーク方向に辿るとき, 真方向出力ポートから辿り始めるか偽方向出力ポートから辿り始めるかにより n_j に到達できるかどうかが変わることを意味する。

[定義 9] 到達影響関係。

PDFG $\mathcal{G}$ において, n_i を *switch-macro* ノードとする。また, 到達影響ノード集合 $\mathcal{N}_{inf}(n_i)$ を次の通り定める。

初期条件 (1) のもと, $\mathcal{N}_{inf}^*(n_i) = \mathcal{N}_{inf}^{*-1}(n_i)$ となるまで (2) を繰り返してできる $\mathcal{N}_{inf}^{**}(n_i)$ を到達影響ノード集合といい, $\mathcal{N}_{inf}(n_i)$ と表す。

- (1) $\mathcal{N}_{inf}^0(n_i) = \{ \}$
- (2) $\mathcal{N}_{inf}^r(n_i) = \mathcal{N}_{inf}^{r-1}(n_i) \cup \mathcal{N}_i$ ($r \geq 1$)

ただし, $\mathcal{N}_i = \{n_i | (n_k \in \mathcal{N}_{inf}^{r-1}(n_i) \vee n_k = n_i) \wedge (n_k \xrightarrow{Rdpnd} n_i)\}$ このとき, $n_j \in \mathcal{N}_{inf}(n_i)$ は n_i へ到達影響であるといい, $n_i \xrightarrow{Ripd} n_j$ と表す。□

到達影響関係は, PDFG を実行させる際, *switch-macro* ノード n_i の真偽判定入力ポートから受信するトークンの値に応じて, ノード n_j が実行されるかどうかが決まることを意味する。

[定義 10] 制御依存関係。

PDFG $\mathcal{G}$ において, $n_i \xrightarrow{Ripd} n_j$ を満たす n_i を *switch-macro* ノード, n_j を *ctrl_dpnd* ノード, n_i の真偽判定入力ポートを x とする。ここで,

$$(x \xrightarrow{C \text{ data_dpnd}} y) \vee (x \xrightarrow{(C \text{ data_dpnd})^*} (D \text{ data_dpnd})^* y)$$

が成立するとき, y は x に制御依存であるといい, $x \xrightarrow{Ctrl} y$ と表す。□

y が x に制御依存であるとは, プログラムを実行させる際, 条件文の条件判定式 x の値に応じて変数 y の値が変わることを意味する。

4.5 依存ノード集合の定義

まず, あるノードから, 依存関係をプログラムの実行方向とは逆の方向に辿って到達できるノード集合を定義する。

[定義 11] 逆方向依存ノード集合。

PDFG $\mathcal{G}$ において, n_i を任意のノード, x を n_i の任意の入力ポートとする。このとき, 逆方向依存ポート集合 $\mathcal{I}_{bk}(n_i)$ を次の通り定める。

初期条件 (1) のもと, $\mathcal{I}_{bk}^*(n_i) = \mathcal{I}_{bk}^{*-1}(n_i)$ となるまで (2) を繰り返してできる $\mathcal{I}_{bk}^*(n_i)$ を逆方向依存ポート集合といい, $\mathcal{I}_{bk}(n_i)$ と表す。また, $\mathcal{I}_{bk}(n_i)$ の要素が属するノードの集合を n_i に対する逆方向依存ノード集合といい, $\mathcal{N}_{bk}(n_i)$ と表す。

- (1) $\mathcal{I}_{bk}^0(n_i) = \{ \}$
- (2) $\mathcal{I}_{bk}^r(n_i) = \mathcal{I}_{bk}^{r-1}(n_i) \cup \mathcal{V}$ ($r \geq 1$)

ただし, $\mathcal{V} = \{y | (z \in \mathcal{I}_{bk}^{r-1}(n_i) \vee z = x) \wedge (y \xrightarrow{Data} z \vee y \xrightarrow{Ctrl} z)\}$ □

次に, あるノードから, 依存関係をプログラムの実行方向と同じ方向に辿って到達できるノード集合を定義する。

[定義 12] 順方向依存ノード集合。

PDFG $\mathcal{G}$ において, n_i を任意のノード, x を n_i の任意の入力ポートとする。このとき, 順方向依存ポート集合 $\mathcal{I}_{fw}(n_i)$ を次の通り定める。

初期条件 (1) のもと, $\mathcal{I}_{fw}^*(n_i) = \mathcal{I}_{fw}^{*-1}(n_i)$ となるまで (2) を繰り返してできる $\mathcal{I}_{fw}^*(n_i)$ を順方向依存ポート集合といい, $\mathcal{I}_{fw}(n_i)$ と表す。また, $\mathcal{I}_{fw}(n_i)$ の要素が属するノードの集合を n_i に対する順方向依存ノード集合といい, $\mathcal{N}_{fw}(n_i)$ と表す。

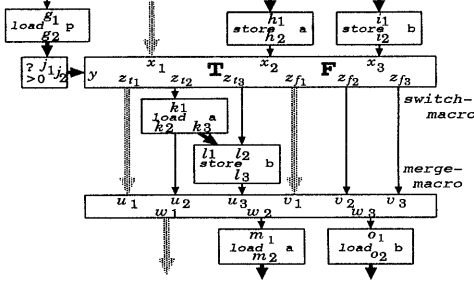
- (1) $\mathcal{I}_{fw}^0(n_i) = \{ \}$
 - (2) $\mathcal{I}_{fw}^r(n_i) = \mathcal{I}_{fw}^{r-1}(n_i) \cup \mathcal{V}$ ($r \geq 1$)
- ただし, $\mathcal{V} = \{y | (z \in \mathcal{I}_{fw}^{r-1}(n_i) \vee z = x) \wedge (z \xrightarrow{Data} y \vee z \xrightarrow{Ctrl} y)\}$ □

4.6 依存関係の例

図4(a)のプログラムに対するPDFGの一部を、図4(b)に示す。図4(b)では、 $h_1 \xrightarrow{Data} k_1$, $h_1 \xrightarrow{Data} l_1$, $y \xrightarrow{Ctrl} l_1$ などの関係がある。

```
void conditional_copy(p_a, p_b, v_p)
int *p_a, *p_b, v_p;
{
    int a, b, p;
    a = *p_a; b = *p_b; p = v_p;
    if (p > 0) {b = a;}
    *p_a = a; *p_b = b;
}
```

(a) 条件つきコピーを行うプログラム
(a) A program for a conditional copy.



(b) (a)のプログラムのPDFGの例
(b) An example of a PDFG corresponding to program (a).

図4 条件分岐文を含むプログラムのPDFGの例
Fig. 4 An example of a program including a conditional statement.

5 一般データフロー計算モデル

一般データフロー計算 (Generalized Dataflow Computing: GDC) モデルは、ノードで実行すべき機能を意味規則としてグラフの構造とは独立に定義し、意味規則に従ってグラフを解釈実行することにより計算を進めるモデルである¹³⁾。ここで、実行対象となる有向グラフを一般データフローグラフ (Generalized Dataflow Graph: GDG) と呼ぶ¹³⁾。

通常のデータフロー計算では、データフローグラフの解釈は一意に定められており、同じグラフを与えると同じ計算を行う。しかし、GDCでは、意味規則を変えることにより、同じGDGに対して種々の解釈を行うことが可能となる。特に、GDCを用いることにより、各種の依存関係を持つ情報をその依存関係に沿って解析し、情報が持つ様々な性質を求めることが容易となる。例えば、GDCは、Infeasible Path (どのような入力を与えても決して実行されない計算経路) 検出^{15), 16), 14), 18)}に適用できる¹³⁾。

5.1 意味規則

GDGに与える意味規則は、ノードの発火条件を指定する発火制約と、ノードで実行すべき機能を指定する実行規則から構成される¹³⁾。そして、一般データフローインタプリタは、指定された意味規則に従い、以下の制御を順次行うことにより、GDGの解釈実行を進める。

(1) ノードの発火制約

発火制約を満たしたノードを発火させる。ノードが発火した際、トークンをアークから取り除き、実行規則の解釈部にトークンを渡す。

(2) トークンの送信制御

実行規則の解釈部が作成したトークンをアークに送信する。

5.2 発火制約

デマンド制約 $c_d \in \{true, \perp\}$, アーク制約 $c_a \in \{\text{アーク属性集合}\}$, 方向制約 $c_r \in \{fw, bk\}$, 発火条件 $c_f \in \{wait, prec\}$ を用いて、発火制約を (c_d, c_a, c_r, c_f) と表す。

以下に、各制約の意味を示す。 $c_d = true$ は、デマンドと呼ばれる信号を出力ポートから受信したノードのみが発火可能となることを表す。 $c_d = \perp$ は、デマンドを受信しなくても発火可能となることを表す。 c_a は、発火判定に用いるアーク属性の最小元を表す。 $c_r = fw$ は、入力ポートに到着するトークンに対して発火判定を行い、作成したトークンを出力ポートに送信することを表す。 $c_r = bk$ は、出力ポートに対して発火判定を行い、トークンを入力ポートに送信することを表す。 $c_f = wait$ は、 c_a と c_r で指定される全ポートにトークンを受信した際、発火させることを表す。 $c_f = prec$ は、あるポートでトークンを受信すると、トークンを待ち合わせずにノードを直ちに発火させることを表す。

5.3 実行規則

実行規則はノードの属性ごとに定める。方向制約 $c_r = fw$ のとき、ノード n に対する実行規則は、 n の各入力ポートより受信したトークンの列から、 n の出力ポートに送信するトークンの列を作成する関数として表す。一方、 $c_r = bk$ のとき、出力ポートから受信したトークン列より入力ポートに送信するトークン列を作成する関数として表す。

5.4 実行制御法

発火制約が $(true, c_a, fw, wait)$ のとき要求駆動型実行¹⁷⁾、 $(\perp, c_a, fw, wait)$ のときデータ駆動型実行¹⁷⁾となる。また、 $(\perp, c_a, fw, prec)$ のときの実行制御法を順方向先行実行と呼び、 $(\perp, c_a, bk, prec)$ のときの実行制御法を逆方向先行実行と呼ぶ。先行実行は、GDGに対する部分的な解析を行う際に有用である。

発火制約と実行規則をGDGに与え、解析したいノードにトークンを送ることによって先行実行は開始される。発火可能なノードと実行中のノードがなくなると、実行は終了する。

5.5 トークンのコンテキスト名

GDCモデルでは、トークンに対して、演算値の他にコンテキスト名¹³⁾を保持させる。コンテキスト名は、通常のデータフロー計算においてタグ、色¹⁷⁾などと呼ばれている、多重処理のためにトークンに付与する実行環境の識別子に対応する。コンテキスト名の集合は最大元 $\top$ と最小元 $\perp$ を含み、要素は半順序関係をなす。ノードの発火制約でトークンの待ち合わせ ($c_f = wait$) を行う際に、 $ct_1 \leq ct_2$ なる関係を持つ ct_1 , ct_2 をコンテキスト名とするトークンが揃うと、 ct_2 でノードを発火させる¹⁹⁾。

6 PDFGの実行

PDFGは、GDCモデルに基づいて実行可能である。求めたい性質に対応した意味規則をPDFGに与え、PDFGを実行させることにより、対応する性質が求まる。

6.1 経路識別子

トークンのコンテキスト名として経路識別子^{15), 16), 14)}を用いると、複数の経路上のトークンを識別し、その経路識別子に対する計算経路に関する性質を求めることができる。

経路識別子の表現方法を以下に示す。プログラム P において、開始文またはある分岐文から、計算経路を実行方向に辿って最初に到達する、分岐文または終了文までの経路を経路断片という。そして、 P における各経路断片に対して、それらを識別する経路断片識別子を付与する。このとき、開始文を含む経路断片に対する経路断片識別子は Id_0^f とする。更に、計算経路 p_i に対する経路断片識別子を、 p_i を構成する経路断片に対応する経路断片識別子の系列により表す。

このとき、経路識別子の集合の要素は木状の半順序関係となる。ここで、ある経路 p_i の経路識別子 Id_i^f と、別の経路 p_j の経路識別子 Id_j^f を考える。そして、 Id_i^f の経路断片識別子の系列が Id_j^f の経路断片識別子の系列に含まれるとき、 Id_i^f は Id_j^f

の先祖であるといひ、 $Id_j^d \prec Id_j^d$ と表す。

6.2 PDFGの実行制御法

PDFGに適用可能な実行規則集合を $\mathcal{R}_0$ で表す。 $\mathcal{R}_0$ は、記述言語の文法規則に従って実行を行う計算実行規則 r_{cal} や、求めたい性質に応じた様々な実行規則を含む。また、初期トークンを tk 、 tk の値を v 、 tk のコンテキスト名を ct 、 tk を送信すべきノードを n としたとき、3つ組 (v, ct, n) を初期トークン条件と呼び、 e_{tk} と表す。そして、 e_{tk} の集合を初期トークン条件集合と呼ぶ。

発火制約 (c_d, c_a, c_r, c_f) 、実行規則集合 $\mathcal{R}_e(\subset \mathcal{R}_0)$ 、初期トークン条件集合をPDFGに与えらると、PDFGの実行が開始される。

6.3 コンテキスト名の更新

トークンのコンテキスト名が Γ の場合、出力トークンのコンテキスト名としては常に Γ を用いる。 Γ でない場合、operationalノードとswitch-macroノードで、発火条件に応じた方法で、以下のようにコンテキスト名を更新する。

発火条件 $c_f=wait$ のとき、operationalノードでは、発火したコンテキスト名をトークンに保持させる。switch-macroノードでは、発火したコンテキスト名に対して送信方向に応じた経路断片識別子を付与したコンテキスト名を、トークンに保持させる。

発火条件 $c_f=prec$ のとき、次に定義するコンテキスト名変換履歴作成規則により与えられるコンテキスト名変換履歴に従い、コンテキスト名を更新する。

[定義13] コンテキスト名変換履歴作成規則。

方向制約 $c_r=fw$ と発火条件 $c_f=wait$ を与えて実行させた際に発火したノードを n 、 n の入力ポートの数を m 、 n が発火した際のコンテキスト名を ct_f 、そのとき n の第 j 番目($1 \leq j \leq m$)の入力ポートから受信したトークンのコンテキスト名を $ct_i(j)$ とすると、 $(m+1)$ つ組 $(ct_i(1), ct_i(2), \dots, ct_i(m), ct_f)$ をコンテキスト名変換履歴と呼び、 h_{ct} と表す。そして、 n に対する h_{ct} の集合をコンテキスト名変換履歴と呼び、 $\mathcal{H}_{ct}(n)$ と表す。

ここで、 n が発火するたびに h_{ct} を $\mathcal{H}_{ct}(n)$ に追加する手続きをコンテキスト名変換履歴作成規則と呼び、 r_h と表す。□
 $c_f=prec$ でコンテキスト名の更新をするためには、まず、発火制約 $(\perp, data_dpnd, fw, wait)$ 、実行規則集合 $\{r_h, r_{cal}\}$ 、トークン初期条件集合 $\{(v, Id_0^f, n_s)\}$ をPDFGに与えてPDFGを実行させ、ノード n に $\mathcal{H}_{ct}(n)$ を作成しておく。

$c_f=prec$ でPDFGを実行させるとき、方向制約 $c_r=fw$ の場合、ノード n の第 j 番目の入力ポートからコンテキスト名 ct^j のトークンを受信して発火した際、 n において、 $ct_i(j)=ct^j$ を満たす $h_{ct}(\in \mathcal{H}_{ct}(n))$ を探し、 h_{ct} における ct_f をトークンのコンテキスト名とする。方向制約 $c_r=bk$ の場合は、ノード n の出力ポートからコンテキスト名 ct^o のトークンを受信して発火した際、 n において、 $ct_f=ct^o$ を満たす $h_{ct}(\in \mathcal{H}_{ct}(n))$ を探し、 h_{ct} における $ct_i(j)$ を、 j 番目の入力ポートに送信するトークンのコンテキスト名とする。

7 PDFGの実行によるスライスの作成

2で述べた各スライスに対し、それぞれの性質を解析するための実行規則を定める。

7.1 スライス作成における基本方針

PDFGにおいて、スライス $(P, I, s, \mathcal{V}, f_i, f_d, f_e)$ を求めるために必要な機能について、スライスの各属性に応じて述べる。

7.1.1 入力属性に対する機能

静的スライス($f_i=st$)を求める場合、PDFGの全アークを用いる。一方、動的スライス($f_i=dy$)を求める場合、特定の入力を与えたときに実行される経路上のアークのみを用いる。

7.1.2 方向属性に対する機能

s の位置で、変数 $u(u \in \mathcal{V})$ の値をloadするノード n_u を着目ノード、 n_u の集合 $\mathcal{N}_a$ を着目ノード集合と呼ぶ。逆方向スライス($f_d=bk$)を求める場合には、 $\mathcal{N}_a$ に対する逆方向着目依存ノード集合 $\mathcal{N}_{bk}^a(\mathcal{N}_a)$ を、順方向スライス($f_d=fw$)を求める場合には、 $\mathcal{N}_a$ に対する順方向着目依存ノード集合 $\mathcal{N}_{fw}^a(\mathcal{N}_a)$ を、以下に示す通りそれぞれ求める。

$$\begin{aligned}\mathcal{N}_{bk}^a(\mathcal{N}_a) &= \{n | n_j \in \mathcal{N}_a, n \in \mathcal{N}_{bk}(n_j)\} \\ \mathcal{N}_{fw}^a(\mathcal{N}_a) &= \{n | n_j \in \mathcal{N}_a, n \in \mathcal{N}_{fw}(n_j)\}\end{aligned}$$

7.1.3 実行可能性属性に対する機能

実行可スライス($f_e=ex$)は、スライスの実行に必要な文をすべて含まなければならない。複数回実行される文がある場合に実行可能なスライスとするには、何回目の実行の際においても必ずその文の計算ができるように注意する必要がある。

例えば、入力 I を与えたとき文 s が j 番目($j < k$)に実行されるようなプログラム P に対して、スライス $(P, I, s^k, \{u\}, dy, bk, ex)$ を考える。ここで、 s の変数 u 、文 t の変数 v をそれぞれ u_s 、 v_t と表し、 s^j における u_s を u_s^j 、 s^k における u_s を u_s^k とそれぞれ表す。更に、 u_s^j は v_t に依存しているが u_s^k は依存していないとする。このとき、実行経路を実行順序とは逆の方向に、 u_s^k から依存関係を辿って到達できる文の集合を求めると、この集合は v_t を含まない。この文集合は、 s の2回目の実行の際には u_s すなわち u_s^2 の計算ができるが、1回目の実行の際には u_s すなわち u_s^1 の計算ができないため、実行可能とはならない。このとき、 u_s^1 の計算に必要な文を求めて上の文集合に付加すれば、実行可能となる。

7.2 スライスの性質と実行規則

スライス属性に対応した実行規則を、表2の通り定める。

表2 スライス作成のための実行規則

属性	スライスの性質	実行規則
f_i	動的(dy)	経路設定規則
f_i	静的(st)/動的(dy)	送信アーク判定規則
f_d	逆方向(bk)/順方向(fw)	依存性解析規則
f_e	実行可(ex)	実行可能化規則

以下では、各規則について、ノードごとに実行動作を定義する。

7.2.1 経路設定規則

経路設定規則は、動的スライス(入力属性 $f_i=dy$)を求める際に用いる経路を定める規則であり、方向制約 $c_r=fw$ のときに有効となる。経路設定規則では、以下で示す送受信履歴と発火コンテキスト名集合とをノードに作成する。

ノード n がコンテキスト名 c で発火したとする。このとき、 c を発火コンテキスト名と呼び、 $C_r(n)$ を n の発火コンテキスト名集合と呼ぶ。そして、 n がswitch-macroノードの場合には、 n の真偽判定入力の値を p としたとき、 (c, p) の組を送信記録と呼び、 h_s と表す。そして、 n に対する h_s の集合を送信履歴と呼び、 $\mathcal{H}_s(n)$ と表す。また、 n がswitch-macroノードの場合には、2組の入力のうちどちらからトークンが到着したかを示す識別子を i としたとき、 (c, i) の組を受信記録と呼び、 h_r と表す。そして、 n に対する h_r の集合を受信履歴と呼び、 $\mathcal{H}_r(n)$ と表す。

[定義14] 経路設定規則。

次に示す処理を行うことを経路設定規則と呼び、 r_{set} と表す。

- switch-macroノード n では送信記録を $\mathcal{H}_s(n)$ に追加する。
- merge-macroノード n では受信記録を $\mathcal{H}_r(n)$ に追加する。
- それ以外のノード n では発火コンテキスト名を $C_r(n)$ に追加する。□

7.2.2 送信アーク判定規則

送信アーク判定規則は、トークンを送信すべきアークを定める規則である。送信アーク判定規則では、動的スライスを求める際、経路設定規則により作成された送受信履歴に従ってアークを選択するため、その履歴を作成したときと同じ実行経路に対するアークを選択することが可能となる。

[定義 15] 送信アーク判定規則。

次に示す処理を行うことを送信アーク判定規則と呼び、 r_{arc} と表す。

- *switch-macro* ノード n では、入力ポートでトークンを受信してコンテキスト名 $c^i (\neq T)$ で発火した場合には、 $c=c^i$ を満たす $h_s=(c,p) \in \mathcal{H}_s(n)$ を探し、 p の値に応じて、真または偽方向出力へのトークン送信を許可する。 $c^i=T$ の場合には、真偽両方向への送信を許可する。
- *merge-macro* ノード n では、出力ポートでトークンを受信してコンテキスト名 $c^o (\neq T)$ で発火した場合には、 $c=c^o$ を満たす $h_r=(c,i) \in \mathcal{H}_r(n)$ を探し、 i の値に応じた入力組へのトークン送信を許可する。 $c^o=T$ の場合には、2つの入力組への送信を許可する。
- それ以外のノードでは何もしない。 □

7.2.3 依存性解析規則

依存性解析規則は、依存関係解析を行う方向を指示する解析方向フラグ $a_d \in \{bk, fw\}$ に従い、依存関係検出ノード集合と呼ぶ解析結果を与える。そして、 $a_d=bk$ なら入力ポート方向に、 $a_d=fw$ なら出力ポート方向に、それぞれ依存関係解析を行う。このとき、依存性解析規則では、データトークンと制御トークンと呼ぶ2種類のトークンを用いる。そして、トークンの値には、これらトークンの識別子を乗せる。この規則をPDFGに与え、ノード n を指定してPDFGを実行させると、 $\mathcal{N}_{bk}^f(n)$ または $\mathcal{N}_{fw}^f(n)$ が求まる。以下に、 $a_d=bk$ の場合の依存性解析規則を定義するが、 $a_d=fw$ の場合も同様に定義できる。

[定義 16] 逆方向依存性解析規則。

PDFG $\mathcal{G}$ において、 $\mathcal{N}^d$ を依存関係検出ノード集合、 n をノード、 x を n の出力ポートとしたとき、 n が x からトークンを受信して発火した際に表3に示す処理を行うことを逆方向依存性解析規則と呼び、 r_{anal} と表す。

表の T の真偽判定のためには、事前に、*switch-macro* ノード n に対する到達影響ノード集合 $\mathcal{N}_{inj}(n)$ を求める実行が必要である。事前の実行を省略して、 $T=true$ として近似的にスライスを求めることも可能である。また、同じポートに対して、値が同じでかつ、コンテキスト名が同じトークンを送信しても、 $\mathcal{N}^d$ に新たに追加されるノードは生じない。このため、*branch* ノードでは、SendXD、SendXCにより、送信したトークンを記録して、同じポートに対して同じトークンを送信しないようにしている。これにより、繰り返し処理を表すサイクルがPDFGにある場合でも、サイクルを有限回実行した後停止することが保証される。

7.2.4 実行可能化規則

実行可能化規則は、調べるべき実行経路を求める手続きである。この規則によりノード n に実行経路を設定して、PDFGを逆方向に実行させると、要求駆動型実行におけるデマンド送信と同様にトークンが送信され、 n の計算に必要なノードが求まる。

[定義 17] 実行可能化規則。

次に示す処理を行うことを実行可能化規則と呼び、 r_{exec} と表す。

- *branch* ノードでは何もしない。
- それ以外のノード n では、コンテキスト名 $c (\neq T)$ のト

表 3 逆方向依存性解析規則

ノード	DTk	CTk
load	Wrt SendD(DTk)	SendD(CTk)
ctrl-dpnd	Wrt SendD(DTk) SendD(CTk)	SendD(CTk)
switch-macro	Wrt SendXD(DTk)	if($T=true$) Wrt SendXD(CTk) SendXC(DTk) if($T=false$) SendXD(CTk)
merge-macro	Wrt SendXD(DTk)	SendXD(CTk)

DTk: データトークン CTk: 制御トークン Wrt: ノードを $\mathcal{N}^d$ に追加する。

SendD(Tk): トークン Tk を、 $y \sqsubseteq x$ を満たす y に送信する。

SendC(Tk): トークン Tk を、 $y \sqsubseteq x$ を満たす y に送信する。

SendXD(Tk): $Tk \notin \mathcal{H}^{tk}(y)$ のとき、 Tk を $\mathcal{H}^{tk}(y)$ に追加して、SendD(Tk) を実行する。

ただし、 $y \sqsubseteq x$ とする。

SendXC(Tk): $Tk \notin \mathcal{H}^{tk}(y)$ のとき、 Tk を $\mathcal{H}^{tk}(y)$ に追加して、SendC(Tk) を実行する。

ただし、 $y \sqsubseteq x$ とする。

T: 式 ($x \sqsubseteq x \wedge u \in \mathcal{N}^d \wedge x \sqsubseteq u$)。

クンを受信した場合、 $C_r(n)$ の要素をコンテキスト名とするトークン集合を作成して、送信する。 $c=T$ の場合、何もしない。 □

7.3 スライス作成のための PDFG の実行

以下に、スライス $(P, I, s, \mathcal{V}, f_i, f_d, f_e)$ を求める手順を示す。ただし、 P のPDFGを $\mathcal{G}$ と表し、データトークンを表す識別子を dt と表す。

(1) 送信アークの設定

$f_i=dy$ のとき、発火制約 $(\perp, \perp, fw, wait)$ 、実行規則集合 $\{r_h, r_{set}, r_{cal}\}$ 、初期トークン条件集合 $\{(I, Id_0^I, n_a)\}$ により、 $\mathcal{G}$ をデータ駆動型実行させる。

これにより、 $\mathcal{G}$ のノード n に対して、コンテキスト名変換履歴 $\mathcal{H}_{ct}(n)$ 、送信履歴 $\mathcal{H}_s(n)$ 、受信履歴 $\mathcal{H}_r(n)$ 、発火コンテキスト名集合 $C_r(n)$ が設定される。

(2) クロージャ・スライスの作成

発火制約 $(\perp, data_dpnd, fd, prec)$ 、実行規則集合 $\{r_{arc}, r_{anal}(f_d)\}$ 、トークン初期条件集合 $\{(dt, ct, n) | n \in \mathcal{N}_a\}$ により、 $\mathcal{G}$ を逆方向または順方向先行実行させる。ただし、 ct は、 $f_i=dy$ の場合には $C_r(n)$ の要素のうち求めたい経路に対応するコンテキスト名を表し、 $f_i=st$ の場合には $ct=T$ である。また、 $\mathcal{N}_a$ は着目ノード集合を表す。

この実行により求まる依存関係検出ノード集合を、着目ノード集合 $\mathcal{N}_a$ に対する着目依存関係検出ノード集合と呼び、 $\mathcal{N}_{cl}^d(\mathcal{N}_a)$ と表す。そして、 $\mathcal{N}_s^d(\mathcal{N}_a)$ に対応する文集合が、求めるべきクロージャ・スライスとなる。

(3) 実行可スライスへの変換

$f_e=ex$ のとき、発火制約 $(\perp, data_dpnd, bk, prec)$ 、実行規則集合 $\{r_{arc}, r_{anal}(bk), r_{exec}\}$ 、初期トークン条件集合 $\{(dt, ct, n) | n \in \mathcal{N}_{cl}^d(\mathcal{N}_a)\}$ により、 $\mathcal{G}$ を逆方向先行実行させる。ただし、 $f_i=dy$ の場合には $ct=\perp$ 、 $f_i=st$ の場合には $ct=T$ とする。

この実行で求まる依存関係検出ノード集合に対応する文集合が、求めるべき実行可スライスとなる。

8 PDFG を用いたスライスの作成例

本章では、図3(a)のプログラム P_1 、図5(a)のプログラム P_2 に対して、PDFGを用いて各種スライスを作成する例を示す。ここで、 P_2 は、次の計算を行うプログラムである。

$$\begin{aligned}
 i_n &= i_{n-1} + 1 & (n \geq 1), & & i_0 &= 0 \\
 a_n &= a_{n-1} + i_{n-1} & (n \geq 1), & & a_0 &= a_{init} \\
 b_n &= b_{n-1} + a_{n-1} & (n \geq 1), & & b_0 &= b_{init} \\
 c_n &= c_{n-1} + i_{n-1} & (n \geq 1), & & c_0 &= c_{init} \\
 d_n &= d_{n-1} + c_{n-1} & (n \geq 1), & & d_0 &= d_{init}
 \end{aligned}$$

関数の引数のうち *p.a, *p.b, *p.c, *p.d は、それぞれ a_{init} , b_{init} , c_{init} , d_{init} に、v.n は添字 n に対応する。

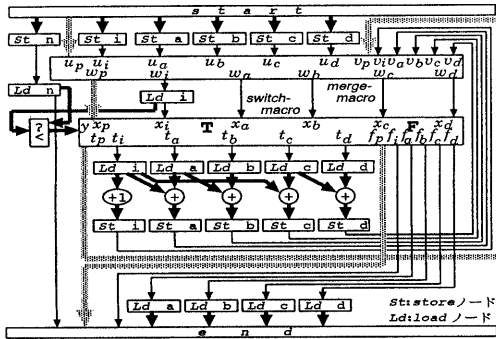
P_2 のプログラムに対するPDFGの例を、図5(b)に示す。

```

void progression(p_a,p_b,p_c,p_d,v_n)
int *p_a,*p_b,*p_c,*p_d,v_n;
{
    int a,b,c,d,n,i;
    a=*p_a; b=*p_b; c=*p_c; d=*p_d;
    n=v_n; i=0;
    while(i < n){
        b+=a; a+=i; d+=c; c+=i; i++;
    }
    *p_a=a; *p_b=b; *p_c=c; *p_d=d;
}

```

(a) 数列を計算するプログラム
(a) A program for progression.



(b) (a) のプログラムの PDFG の例
(b) An example of a PDFG corresponding to program (a).

図 5 サンプルプログラム (2)
Fig. 5 A sample program (2).

表 4 図 3(a), 図 5(a) に対するスライス

文/スライス	S^a	S^b	S^c	S^d	S^e	文/スライス	S^f	S^g
$a=*p_a$	○	×	○	×	●	$a=*p_a$		
$b=*p_b$	○	○	○	○	○	$b=*p_b$		
$c=*p_c$	○	×	×	○	●	$c=*p_c$	●	●
$n=v_n$	○	○	○	○	○	$d=*p_d$		
$w(n-1)$	○	○	○	○	○	$n=v_n$		●
$r=c$	○	×	×	×	×	$i=0$		○
$c=b$	○	○	○	×	○	$w(i>n)$	○	○
$b=a$	○	×	○	○	○	$b+=a$		
$a=r$	○	×	×	×	×	$d+=c$	○	○
$*p_a=a$				×	×	$c+=i$	○	○
$*p_b=b$				○	×	$i++$		●
$*p_c=c$	●	●	●	×	○	$*p_a=a$		
						$*p_b=b$		
						$*p_c=c$	○	○
						$*p_d=d$	○	○

ただし, $w()$ は while() を表す.

上は図 3(a) を, 右は図 5(a) を示す.

スライス $(P_1, \perp, s_{12}, \{c\}, st, bk, cl)$ を S^a , スライス $(P_1, (I_0, v_n=1), s_{12}, \{c\}, dy, bk, cl)$ を S^b , スライス $(P_1, (I_0, v_n=2), s_{12}, \{c\}, dy, bk, ex)$ を S^c と表す. ただし, s_{12} は文 $*p_c=c$; I_0 は $(*p_a=a_0, *p_b=b_0, *p_c=c_0)$ を表す. また, スライス $(P_2, \perp, s_3, \{c\}, st, fw, cl)$ を S^d , スライス $(P_2, \perp, s_3, \{c\}, st, fw, ex)$ を S^e と表す. ただし, s_3 は文 $c=*p_c$; s_1 は文 $a=*p_a$; s_2 は文 $b=*p_b$ を表す. さらに, スライス $(P_1, (I_0, v_n=1), s_1, \{a\}, dy, fw, cl)$ を S^f , スライス $(P_1, (I_0, v_n=2), s_1, \{a\}, dy, fw, ex)$ を S^g と表す. ただし, s_1 は文 $a=*p_a$ を表す.

S^a から S^g に対するスライスの作成結果を表 4 に示す. 表において, ●印は着目文を示し, ○印は, 7.3 の (2) におけるクロージャ・スライス作成のための実行により, スライスの要素であると判定される文を示す. そして, ×印は, 送信アーク判定規則のために, データトークンや制御トークンが到達しないノードに対応する文を示す. 更に, ●印は, 7.3 の (3) における実行可能スライスへの変換のための実行により, スライスに付加される文を示す.

9 むすび

統一的なフレームワークにより様々なスライスを並列に作成する手法を提案した. 本稿では, 従来のスライスを 3 種類の

属性に従い分類し, 各属性に対応する意味規則を定めた. そして, 求めたいスライスに応じた意味規則を PDFG のノードに与え, GDC モデルに基づき PDFG を解釈実行させることにより, 各種スライスを作成する方法について述べた. これにより, 既存のスライスおよび, 従来考慮されていなかったスライスを求めることが可能となった.

本稿では, 代入文, 条件分岐文, 繰り返し処理の構文を備え, 基本データ型の局所変数のみを持つプログラムを対象として, スライスの作成法を議論した.

今後, 対象プログラムの領域を拡大する方法について検討を進める予定である. また, スライスの精度の向上や, スライスの一般化により定義された新しい種類のスライスに対する応用について, 検討を進める予定である.

謝辞 日ごろ御指導御討論頂く後藤滋樹部長, 伊藤正樹リーダーはじめソフトウェア基礎技術研究部の皆様に感謝致します.

参考文献

- Weiser, M.: Program Slicing, *IEEE Trans. Software Engineering*, Vol. SE-10, No.4, pp.352-357 (Jul. 1984).
- Ottenstein, K.J. and Ottenstein, L.M.: The Program Dependence Graph in a Software Development Environment, *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symp. on Practical Software Development Environments*, *SIGPLAN Notices*, Vol.19, No.5, pp.177-184 (May 1984).
- Horwitz, S., Prins, J. and Reps, T.: Integrating Noninterfering Versions of Programs, *ACM Trans. Programming Languages and Systems*, Vol.11, No.3, pp.345-387 (Jul. 1989).
- Horwitz, S., Reps, T. and Binkley, D.: Interprocedural Slicing Using Dependence Graphs, *ACM Trans. Programming Languages and Systems*, Vol.12, No.1, pp.26-60 (Jan. 1990).
- Korel, B. and Laski, J.: Dynamic program slicing, *Information Processing Letters*, Vol.29, pp.155-163 (Oct. 1988).
- Korel, B.: PELAS—Program Error-Locating Assistant System, *IEEE Trans. Software Engineering*, Vol.14, No.9, pp.1253-1260 (Sep. 1988).
- Agrawal, H. and Horgan, J.R.: Dynamic Program Slicing, *ACM SIGPLAN Notices*, Vol.25, No.6, pp.246-256 (Jun. 1990).
- 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, *情報処理論文誌*, Vol.33, No.4, pp.501-511 (Apr. 1992).
- Korel, B. and Laski, J.: Dynamic Slicing of Computer Programs, *J. Systems Software*, Vol.13, pp.187-195 (1990).
- Venkatesh, G.A.: The Semantic Approach to Program Slicing, *Proc. of the ACM SIGPLAN'91 Conf. on Programming Language Design and Implementation*, *SIGPLAN Notices*, Vol.26, No.6, pp.107-119 (Jun.1991).
- 高橋直久: ソフトウェア・リエンジニアリングにおける情報の構造と変換, *情報処理学会情報システム研究会*, 42-3, pp.27-36 (Jan. 1993).
- 直井邦彰, 高橋直久: 経路依存フローグラフを用いたプログラム・スライシング, *情報処理学会第 46 回全国大会*, 6E-7 (Mar. 1993).
- 高橋直久, 鈴木英明, 直井邦彰, 福田晴元: データフロー計算の一般化 — ソフトウェア・リエンジニアリングにおける複雑な情報の構造化と理解を目指して —, *ソフトウェア科学会第 10 回全国大会*, C9-2, pp.361-364 (Jun. 1993).
- 直井邦彰, 高橋直久: 経路依存フローグラフを用いたプログラム解析, *NTT R&D*, Vol. 42, No. 8, pp. 1007-1016 (Aug. 1993).
- 直井邦彰, 高橋直久: プレスブルガー算術を用いた Infeasible Path 検出法, *電子情報通信学会知能ソフトウェア工学研究会*, KBSE92-38, pp.57-64 (Nov. 1992).
- 直井邦彰, 高橋直久: 経路依存フローグラフを用いた Infeasible Path 検出法, *電子情報通信学会論文誌*, Vol. J76-D-I, No.8, pp.429-439 (Aug. 1993).
- 雨宮真人: データフロー・アーキテクチャについて, *コンピュータソフトウェア*, Vol.1, No.1, pp.42-63 (Apr. 1984).
- 直井邦彰, 高橋直久: 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, *情報処理学会第 45 回全国大会*, 6T-1 (Oct. 1992).
- Ono, S., Takahashi, N. and Amamiya, M.: Partial Computation with a Dataflow Machine, *Lecture Notes in Computer Science 220, RIMS Symposium on Software Science and Engineering II*, Springer-Verlag, pp.87-113 (1985).