

特別研究報告

題目

関数呼び出し履歴を利用した
プログラムスライス抽出技法の提案と実現

指導教官

井上克郎 教授

報告者

地平 稔

平成10年2月19日

大阪大学 基礎工学部 情報工学科

関数呼び出し履歴を利用した
プログラムスライス抽出技法の提案と実現

地平 稔

内容梗概

プログラムスライシング技術はプログラム内の命令の間の依存関係を明らかにする技術である。デバッグ時にプログラムスライシング技術を用いることで、デバッグの対象をプログラム全体ではなく、その一部分へと狭めることができ、デバッグの効率向上が期待できる。

プログラムスライスには、プログラム内のある命令の実行に影響を与える可能性のある全ての命令を抽出するスタティックスライスと、プログラムにある特定の入力を与えて実行して得られる実行系列から、ある実行時点に影響を与える命令を抽出するダイナミックスライスがある。スタティックスライスは実行される可能性のある全ての命令について考えるため、プログラムスライスとして抽出される命令数がダイナミックスライスよりも大きくなる。一方、ダイナミックスライスは実行系列の保存のために多くの時間および多くの記憶領域が必要となる。

本研究で提案するプログラムスライス抽出技法は、動的な情報をスタティックスライスに組み合わせることで、小さな手間でスタティックスライスの一部を抽出する技法である。この技法では、関数の呼び出し履歴と実行依存関係から、実行されていない文の集合を特定することで、従来のスタティックスライスより小さな文の集合を抽出できる。

主な用語

プログラムスライス (Program Slice)

スタティックスライス (Static Slice)

ダイナミックスライス (Dynamic Slice)

関数呼び出し履歴

目次	
1 まえがき	3
2 プログラムスライス	5
2.1 プログラムスライシング技術の概要	5
2.2 入力言語	5
2.3 スタティックスライス	5
2.3.1 スタティックスライスを計算するアルゴリズム	5
2.3.2 スタティックスライスの計算	7
2.4 ダイナミックスライス	12
2.4.1 ダイナミックスライスを計算するアルゴリズム	12
2.4.2 ダイナミックスライスの計算	14
2.5 関数呼び出し履歴を利用したプログラムスライス	18
2.5.1 関数呼び出し履歴を利用したプログラムスライスを計算するアルゴリズム	18
2.5.2 関数呼び出し履歴を利用したプログラムスライスの計算	19
3 プログラム文の依存関係を利用した開発システム	21
3.1 システム構成	21
3.2 システム利用	22
4 システムに拡張した機能	23
4.1 目的	23
4.2 操作法	23
4.3 内部仕様	25
4.4 実行例	25
5 考察	27
6 あとがき	29
謝辞	30
参考文献	31
付録	33

1 まえがき

ソフトウェアの大規模化に伴い、その複雑さも増している。そのためプログラムの開発や保守の段階で、プログラム全体を対象としてデバッグ作業を行なうのは容易ではない。このような場合に、フォールトに関係があると思われる部分をプログラムから抽出する技術としてプログラムスライスがある。

プログラムスライスの種類として、プログラム内のある命令の実行に影響を与える可能性のある全ての命令を抽出するスタティックスライスと、プログラムにある特定の入力を与えて実行して得られる実行系列から、ある実行時点に影響を与える命令を抽出するダイナミックスライスがある。スタティックスライスは実行される可能性のある全ての命令について考えるため、プログラムスライスとして抽出される命令数がダイナミックスライスよりも大きくなる。一方、ダイナミックスライスは実行系列の保存のために多くの時間および記憶領域が必要となる。このようにスタティックスライス、ダイナミックスライスにはそれぞれ欠点がある。

本研究で提案するプログラムスライス抽出技法は、動的な情報をスタティックスライスに組み合わせることで、小さな手間でスタティックスライスの一部を抽出する技法である。この技法では関数の呼び出し履歴を、動的な情報として用いる。関数の呼び出し履歴と実行依存関係から、実行されていない文の集合を特定することで、従来のスタティックスライスより小さな文の集合を抽出できる。

これまでにプログラム文の依存関係を利用した開発支援システムを作成した。これは、スタティックスライスおよびダイナミックスライスを利用した開発支援システムである。本研究では、このシステムに対して関数呼び出し履歴を利用したプログラムスライスを抽出する機能を追加した。

本システムが扱う言語は大阪大学基礎工学部情報工学科三年時の学生実験で用いられる言語を元にした Pascal のサブセットである。この言語は演習用のため簡単化されてはいるものの基本的な文の構造は全て含んでいるので、将来実際に使用されている手続き型言語（例えば C や Pascal など）に対して、この手法を用いて開発支援システムを作成することは可能である。

本論文ではこれ以降,2節でプログラムスライスの概要及びスタティックスライス,ダイナミックスライス,関数呼び出し履歴を利用したプログラムスライスの計算方法,3節でプログラム文の依存関係を利用した開発支援システムの説明を,4節で関数呼び出し履歴を利用したプログラムスライスを計算するスライサの説明を,5節で考察を述べる.

2 プログラムスライス

2.1 プログラムスライシング技術の概要

プログラムスライシング (Program Slicing) 技術 [16] とは、プログラム中のある文 s におけるある変数 v に対して v の値に影響を与える全ての文、もしくは、 s における変数 w の定義に対して w が影響を与える全ての文をプログラムから抽出する技術で、その結果取り出された文の集合をプログラムスライス (Program Slice) または単にスライス (Slice) と呼ぶ。スライスのうち、全ての可能な入力データに関するスライスをスタティックスライス (Static Slice) と呼び、ある特定の入力のもとで実行された文を並べた実行系列を解析して求められたスライスをダイナミックスライス (Dynamic Slice) と呼ぶ。

2.2 入力言語

大阪大学基礎工学部情報工学科 3 年時の演習で用いられる言語を元にした Pascal のサブセットを入力言語とする。この言語には文として条件文 (if 文)、代入文、繰り返し文 (while 文)、入力文 (readln 文)、出力文 (writeln 文)、手続き呼び出し文、複合文 (begin-end) がある。変数の型としてはスカラ型のみでポインタ型はない。プログラムは、大域変数宣言、手続き (関数) 定義、メインプログラムからなり、ブロック構造はない。手続き内では内部で宣言された局所変数と仮引数変数および大域変数のみが参照可能で、他の手続き内の局所変数は参照できない。手続きは、自己再帰的および相互再帰的に定義可能であり、その引数は、値渡しで扱われる。

2.3 スタティックスライス

2.3.1 スタティックスライスを計算するアルゴリズム

スタティックスライスの計算方法として [12] のアルゴリズムを用いる。このアルゴリズムは [8, 15] を参考にしてプログラム中の文の依存関係を調べ、それを元にプログラム依存グラフ (Program Dependence Graph 略して PDG) を作成し、その辺をたどることによりスタティックスライスを計算する。PDG は有向グラフであるので、そのたどり方にも二種類あり、有向辺を順方向にたどっていくことにより計算したスタティックスライスを Forward Slice、逆方向にたどっていくことにより計算したスタティックスライスを Backward Slice と呼ぶ。

- 諸定義

PDGはプログラム内の文の依存関係を表すグラフである。PDGの節点はプログラム中の各文及びif文やwhile文の条件判定部分を表し、辺は変数の影響を伝えるデータ依存(Data Dependence, 略してDD)関係及び条件文や繰り返し文の制御の影響を伝える制御依存(Control Dependence, 略してCD)関係を表す。

DDは、各頂点の到達定義集合(Reaching Definitions, 略してRD)を求めることによって得られる。PDG上でのある頂点 t のRDとは、変数 v と頂点 s との組 $\langle v, s \rangle$ の集合である。これは、

- (a) プログラム中の文 s で変数 v を定義している。
- (b) プログラム中の二つの文 s と t の間に v を必ず定義するような文がない。

ことを示している。 t のRDに $\langle v, s \rangle$ が含まれ、かつ t が v を参照する時、 s から t へのDD関係があるという。

また、ある条件判定部分 s の結果により文 t の実行の有無が決定される時、 s と t の間にCDが存在するものとする。すなわちCDはif文やwhile文の条件判定部分からそれらの内部ブロックに属する文への影響であり、これはプログラムを解析すれば容易に求められる。

一般にプログラムには複数の手続きが定義されている。各手続き間には引数や大域変数を通じてDD関係が生じる。これらのDD関係を表すために、PDGにプログラム中の文とは直接対応しない節点(中継節点と呼ぶ)を用意する。

PDGの作成は、プログラムを解析し、プログラムの各文をPDGの節点に切りわけ、プログラム中の各文におけるRDを求め、それをもとにしてPDGの各辺を生成することによって行なわれる。詳細は、[15]を参照されたい。

上記の方法で生成されたPDGを G 、 G に含まれる全ての辺の集合を E とすると、プログラム内の文 S の変数 v に関するスライスを表す節点の集合 V は以下のようにして計算される。

1. $V \leftarrow \{n | n \text{ は } S \text{ に対応する節点.}\}$
2. $N \leftarrow \{n | \langle v, n \rangle \in RD_{in}(S)\}$
3. $N \neq \phi$ の間以下の動作を繰り返す。

- $l \in N$ を一つ選ぶ.
- $N \leftarrow N - \{l\}$
- $V \leftarrow V \cup \{l\}$
- $N \leftarrow N \cup \{k \mid (k \notin V) \wedge (k \xrightarrow{*} l \in E \vee k \dashrightarrow l) \wedge (k \text{ が } g\text{-in でない}) \wedge (k \text{ が } para\text{-in 節点でない})\}$ (ただし $k \xrightarrow{*} l$ は任意の変数の k から l への DD 関係辺を示す.)
- k が $para\text{-in}$ 節点でも $para\text{-in}$ 節点でもないなら
 $N \leftarrow N \cup \{k \mid (k \notin V) \wedge (k \xrightarrow{*} l \in E \vee k \dashrightarrow l)\}$ (ただし $k \xrightarrow{*} l$ は任意の変数の k から l への DD 関係辺を示す.)

2.3.2 スタティックスライスの計算

上記の手順に従って 図1 のプログラムにたいしてスタティックスライスを実際に計算する. その際, 作成された PDG を 図2, そのスタティックスライスを 図3, 図4 に示し, 比較のため 図3, 図4 には元のプログラムを一緒につけておく.

図3 は 図1 の 36 行目の文 `writeln(g)` に対応する PDG の節点の変数 g に関する Backward Slice である. このスライスには変数 1 を求めるための関数 `1cm` や関数 `1cm` を呼び出す文, 変数 1 を出力する文などはスタティックスライスに含まれていない.

図4 は, 図1 の 24 行目の文の `w:=m mod n` に対応する PDG の節点の変数 w に関する Forward Slice である. このスライスには手続き `swap` や 32 行目の入力文 `readln(x,y)` などはスライスに含まれていない.

```

1 program euclid(input,output);
2   var    x,y,g,l:integer;
3 function gcd(m,n:integer):integer;forward;
4 procedure swap(var a,b:integer);
5   var    temp:integer;
6   begin
7     temp:=a;
8     a:=b;
9     b:=temp;
10  end;
11 function lcm(a,b:integer):integer;
12   var    c:integer;
13   begin
14     c:=gcd(a,b);
15     lcm:=(a div c)*(b div c)*c
16   end;
17 function gcd;
18   var    w:integer;
19   begin
20     if m < n then begin
21       swap(m,n);
22     end;
23     while n < > 0 do begin
24       w:=m mod n;
25       m:=n;
26       n:=w;
27     end;
28     gcd:=m;
29   end;
30 begin
31   writeln('Input x and y');
32   readln(x,y);
33   writeln('x=',x,' y=',y);
34   g:=gcd(x,y);
35   l:=lcm(x,y);
36   writeln('gcd=',g);
37   writeln('lcm=',l);
38 end.

```

図 1: プログラム

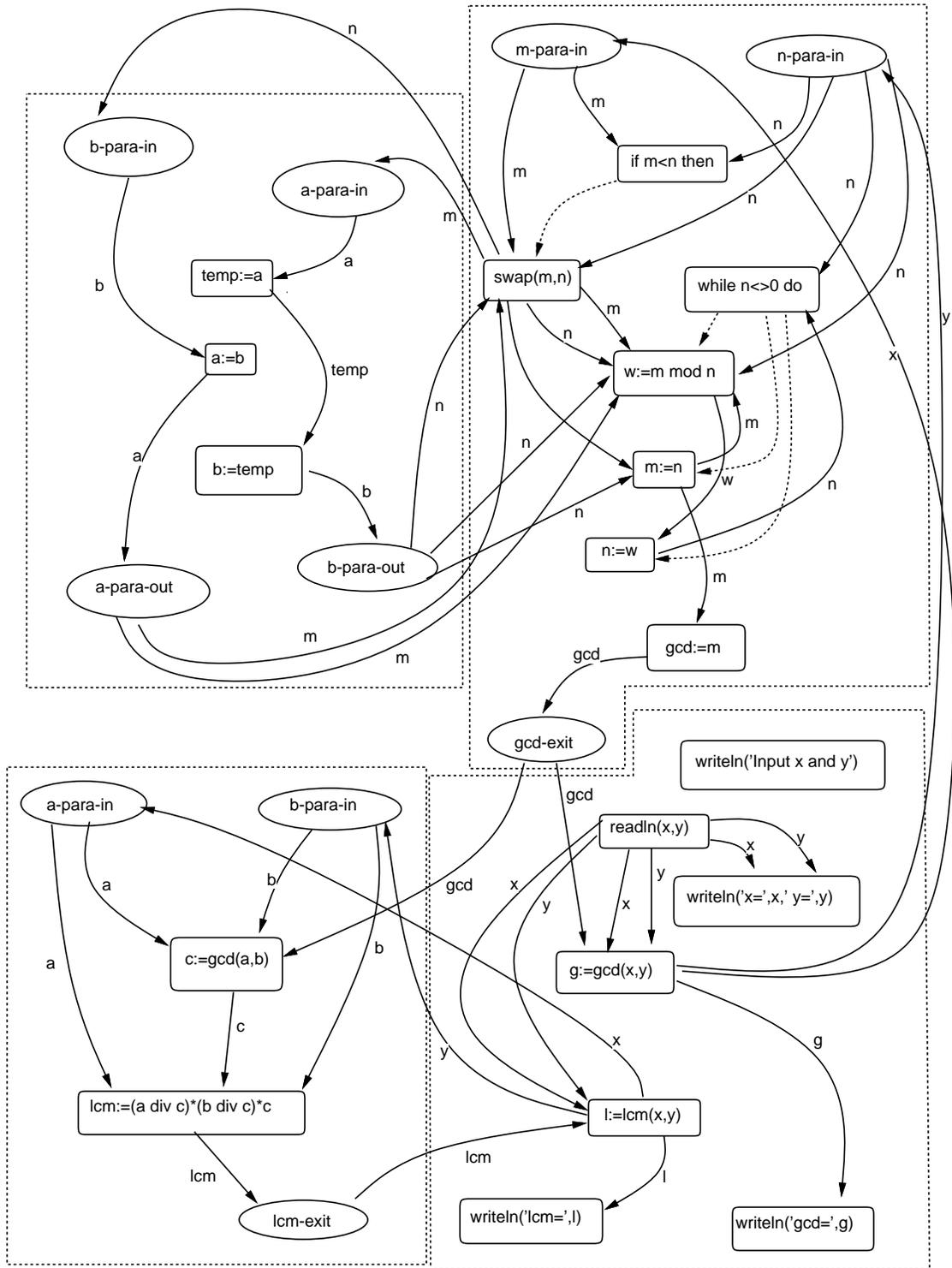


図 2: PDG の例

<pre> 1 program euclid(input,output); 2 var x,y,g,l:integer; 3 function gcd(m,n:integer):integer;forward; 4 procedure swap(var a,b:integer); 5 var temp:integer; 6 begin 7 temp:=a; 8 a:=b; 9 b:=temp; 10 end; 11 function lcm(a,b:integer):integer; 12 var c:integer; 13 begin 14 c:=gcd(a,b); 15 lcm:=(a div c)*(b div c)*c 16 end; 17 function gcd; 18 var w:integer; 19 begin 20 if m < n then begin 21 swap(m,n); 22 end; 23 while n < > 0 do begin 24 w:=m mod n; 25 m:=n; 26 n:=w; 27 end; 28 gcd:=m; 29 end; 30 begin 31 writeln('Input x and y'); 32 readln(x,y); 33 writeln('x=',x,' y=',y); 34 g:=gcd(x,y); 35 l:=lcm(x,y); 36 writeln('gcd=',g); 37 writeln('lcm=',l); 38 end.</pre>	<pre> 1 program euclid(input,output); 2 var x,y,g,l:integer; 3 function gcd(m,n:integer):integer;forward; 4 procedure swap(var a,b:integer); 5 var temp:integer; 6 begin 7 temp:=a; 8 a:=b; 9 b:=temp; 10 end; 17 function gcd; 18 var w:integer; 19 begin 20 if m < n then begin 21 swap(m,n); 22 end; 23 while n < > 0 do begin 24 w:=m mod n; 25 m:=n; 26 n:=w; 27 end; 28 gcd:=m; 29 end; 30 begin 32 readln(x,y); 34 g:=gcd(x,y); 36 writeln('gcd=',g); 38 end.</pre>
--	--

図 3: 元のプログラムとその Backward Slice の比較

<pre> 1 program euclid(input,output); 2 var x,y,g,l:integer; 3 function gcd(m,n:integer):integer;forward; 4 procedure swap(var a,b:integer); 5 var temp:integer; 6 begin 7 temp:=a; 8 a:=b; 9 b:=temp; 10 end; 11 function lcm(a,b:integer):integer; 12 var c:integer; 13 begin 14 c:=gcd(a,b); 15 lcm:=(a div c)*(b div c)*c 16 end; 17 function gcd; 18 var w:integer; 19 begin 20 if m < n then begin 21 swap(m,n); 22 end; 23 while n <> 0 do begin 24 w:=m mod n; 25 m:=n; 26 n:=w; 27 end; 28 gcd:=m; 29 end; 30 begin 31 writeln('Input x and y'); 32 readln(x,y); 33 writeln('x=',x,' y=',y); 34 g:=gcd(x,y); 35 l:=lcm(x,y); 36 writeln('gcd=',g); 37 writeln('lcm=',l); 38 end.</pre>	<pre> 1 program euclid(input,output); 2 var x,y,g,l:integer; 3 function gcd(m,n:integer):integer;forward 11 function lcm(a,b:integer):integer; 12 var c:integer; 13 begin 14 c:=gcd(a,b); 15 lcm:=(a div c)*(b div c)*c 16 end; 17 function gcd; 18 var w:integer; 19 begin 23 while n<>0 do begin 24 w:=m mod n; 25 m:=n; 26 n:=w; 27 end; 28 gcd:=m; 29 end; 30 begin 34 g:=gcd(x,y); 35 l:=lcm(x,y); 36 writeln('gcd=',g); 37 writeln('lcm=',l); 38 end.</pre>
---	---

図 4: 元のプログラムとその Forward Slice の比較

2.4 ダイナミックスライス

2.4.1 ダイナミックスライスを計算するアルゴリズム

入力を与えてプログラムを実行した場合、実行された文の列を実行系列と呼ぶ。実行系列で p 番目に文の実行が行われた時点を実行時点 p と呼ぶ。ダイナミックスライス [2, 10] を計算するために必要となる動的依存関係情報を実行系列に与える。動的依存関係を以下の様に定義する。

1. データ依存関係 (Definition-Use, DU)

$p = \text{Def}(q, w)$ とは、実行時点 q より前で最後に変数 w を定義した実行時点が p であることを表す。最後に定義されたとは、 $p < r < q$ であるような全ての実行系列 r において、変数 w が定義されていないことをいう。また、実行時点 p において実行された文を $\text{Ins}(p)$ で表し、実行時点 p に対して、 $\text{Def}(p)$ は実行時点 p における文 $\text{Ins}(p)$ の実行により定義された変数、 $\text{Use}(p)$ は実行時点 p における文 $\text{Ins}(p)$ の実行により使用された変数の集合を表す。実行時点 p から実行時点 $q (p < q)$ に対してデータ依存関係 $\text{DU}(p, q)$ があるとは、ある変数 $w \in \text{Use}(q)$ が存在して、 $p = \text{Def}(q, w)$ の場合をいう。データ依存関係 $\text{DU}(p, q)$ は、実行時点 p で設定したある変数 w の値が実行時点 q で参照されたことを示している。変数 $w \in \text{Use}(q)$ に関してデータ依存関係があることを明示する場合には、 $\text{DU}_w(p, q)$ と表すこととする。

2. 制御依存関係 (Test-Control, TC)

まず、命令 t に対して、命令の集合 $\text{CtlExec}(t)$ を次のように定義する。

$$\text{CtlExec}(t) = \{ \text{命令 } s \mid \text{CD}(s, t) \}$$

ただし、命令 t がループ命令の場合には命令 t も $\text{CtlExec}(t)$ に含める。

実行時点 p から実行時点 $q (p < q)$ に対して制御依存関係 $\text{TC}(p, q)$ があるとは、

$$p = \max \{ \text{実行時点 } i \mid i < q \text{ かつ } \text{Ins}(i) \in \text{CtlExec}(\text{Ins}(q)) \}$$

の場合をいう。

制御依存関係 $\text{TC}(p, q)$ は、実行時点 q において命令 $\text{Ins}(q)$ が実行されたことは、実行時

点 p において実行された分岐命令あるいはループ命令 $\text{Ins}(p)$ の制御移行に依存していたことを示している。

また、実行系列 q における命令 $\text{Ins}(q)$ が、関数、手続き内にある命令であるなら、その関数、手続きを呼び出した実行時点 q との間にも制御依存関係 $\text{TC}(p, q)$ があるという。このとき実行時点 q における命令 $\text{Ins}(q)$ は、関数、手続き呼び出し文である。

上記の依存関係を元に以下に示す依存関係情報を実行系列に与える。変数 v と実行時点 p との組 r を $\langle v, p \rangle$ と表す。ある実行時点 p の参照変数とその変数が定義された実行時点の組の集合を $\text{VarREF}(p)$ と書き、これを以下のように定義する。

$$\text{VarREF}(p) \equiv \{ \langle v, q \rangle \mid \text{DU}v(q, p) \}$$

また、ある実行時点 p と制御依存関係がある実行時点 q の集合を $\text{Control}(p)$ と表す。

次にこれまでの手順により依存情報を与えられた実行系列を元にプログラム P のダイナミックスライスを計算する。まず、どの実行時点のどの変数に関してダイナミックスライスを計算するかを決定する。その実行時点をダイナミックスライスのスライシング基準といひ $C = (x, r, V)$ と表す。 x, r, V は以下のとおりである。

- x はプログラム P に与えた入力。
- r はプログラム P に入力 x を与えたときの実行系列における実行時点。
- V はプログラム P 内の変数の部分集合。

プログラム P のスライシング基準 $C = (x, r, V)$ に関するダイナミックスライスは以下の手順により求められる。

1. $\text{DS} \leftarrow \phi$

$\text{CalcDsObj} \leftarrow \phi$

CalcDsObj をダイナミックスライスを計算する対象となる実行時点の集合とする。

2. $DS \leftarrow \{Ins(r)\}$

$CalcDsObj \leftarrow \{\text{実行時点 } q \mid \text{ある変数 } v \in V \text{ に対して } q = Def(r, v)\}$

3. $p \in CalcDsObj$

$CalcDsObj \leftarrow \{\text{実行時点 } q \mid VarREF(p) \cup Control(p)\} \cup CalcDsObj$

$DS \leftarrow DS \cup \{Ins(p)\}$

$CalcDsObj \leftarrow CalcDsObj - \{p\}$

4. $CalcDsObj$ が空集合でないなら 3 を行う。 $CalcDsObj$ が空集合なら DS がプログラム P のスライシング基準 $C = (x, r, V)$ に関するダイナミックスライスとなる。

2.4.2 ダイナミックスライスの計算

ここで実際に 図5 のプログラムに異なる入力を与えダイナミックスライスを計算する。このプログラムは、5 個の数字を読み込みその中の最大値及び最小値を出力するプログラムである。

- 入力に $a[1]=1, a[2]=2, a[3]=4, a[4]=3, a[5]=5$ を与えて実行したときの実行系列を 図7 に示す。上記の手順にしたがって 図7 の実行時点43の変数 min に関するダイナミックスライスを計算すると 図5 の左のダイナミックスライスが得られる。
- 入力に $a[1]=5, a[2]=4, a[3]=2, a[4]=3, a[5]=1$ を与えて実行したときの実行系列を 図8 に示す。上記の手順にしたがって 図7 の実行時点43の変数 min に関するダイナミックスライスを計算すると 図5 の右のダイナミックスライスが得られる。

```

1 program max_min(input,output);
2 var i,max,min:integer;
3   a:array[1..5] of integer;
4 begin
5   i:=1;
6   writeln('Input 5 numbers');
7   while i<=5 do
8     begin
9       readln(a[i]);
10      i:=i+1
11    end;
12   min:=a[1];
13   max:=a[1];
14   i:=2;
15   while i<=5 do
16     begin
17       if min>a[i] then
18         min:=a[i] ;
19       if max<a[i] then
20         max:=a[i] ;
21       i:=i+1
22     end;
23   writeln('MAX=',max);
24   writeln('MIN=',min)
25 end.

```

図 5: プログラム

<pre> 1 program max_min(input,output); 2 var i,max,min:integer; 3 a:array[1..5] of integer; 4 begin 5 i:=1; 7 while i<=5 do 8 begin 9 readln(a[i]); 11 end; 12 min:=a[1]; 24 writeln('MIN=',min) 25 end. </pre>	<pre> 1 program max_min(input,output); 2 var i,max,min:integer; 3 a:array[1..5] of integer; 4 begin 5 i:=1; 7 while i<=5 do 8 begin 9 readln(a[i]); 10 i:=i+1 11 end; 12 min:=a[1]; 14 i:=2; 15 while i<=5 do 16 begin 17 if min>a[i] then 18 min:=a[i] ; 21 i:=i+1 22 end; 24 writeln('MIN=',min) 25 end. </pre>
--	---

図 6: 異なる入力によるダイナミックスライス

```

1   i:=1                               {i=1}
2   writeln('Input 5 numbers')
3   while i<=5 do                       {1<=5}
4     readln(a[i])                      {a[1]=1}
5     i:=i+1                             {i=2}
6   while i<=5 do                       {2<=5}
7     readln(a[i])                      {a[2]=2}
8     i:=i+1                             {i=3}
9   while i<=5 do                       {3<=5}
10    readln(a[i])                      {a[3]=4}
11    i:=i+1                             {i=4}
12  while i<=5 do                       {4<=5}
13    readln(a[i])                      {a[4]=3}
14    i:=i+1                             {i=5}
15  while i<=5 do                       {5<=5}
16    readln(a[i])                      {a[5]=5}
17    i:=i+1                             {i=6}
18  while i<=5 do                       {6<=5}
19    min:=a[1]                          {min=1}
20    max:=a[1]                          {max=1}
21    i:=2                                {i=2}
22  while i<=5 do                       {5<=5}
23    if min>a[i] then                   {1>2}
24    if max<a[i] then                   {1<2}
25      max:=a[i]                        {max=2}
26    i:=i+1                             {i=3}
27  while i<=5 do                       {3<=5}
28    if min>a[i] then                   {1>4}
29    if max<a[i] then                   {2<4}
30      max:=a[i]                        {max=4}
31    i:=i+1                             {i=4}
32  while i<=5 do                       {4<=5}
33    if min>a[i] then                   {1>3}
34    if max<a[i] then                   {4<3}
35    i:=i+1                             {i=5}
36  while i<=5 do                       {5<=5}
37    if min>a[i] then                   {1>5}
38    if max<a[i] then                   {4<5}
39      max:=a[i]                        {max=5}
40    i:=i+1                             {i=6}
41  while i<=5 do                       {6<=5}
42    writeln('MAX=',max)                {writeln(5)}
43    writeln('MIN=',min)               {writeln(1)}

```

图 7: 実行系列

```

1   i:=1                                {i=1}
2   writeln('Input 5 numbers')
3   while i<=5 do                        {1<=5}
4     readln(a[i])                       {a[1]=5}
5     i:=i+1                              {i=2}
6   while i<=5 do                        {2<=5}
7     readln(a[i])                       {a[2]=4}
8     i:=i+1                              {i=3}
9   while i<=5 do                        {3<=5}
10    readln(a[i])                       {a[3]=2}
11    i:=i+1                              {i=4}
12  while i<=5 do                        {4<=5}
13    readln(a[i])                       {a[4]=3}
14    i:=i+1                              {i=5}
15  while i<=5 do                        {5<=5}
16    readln(a[i])                       {a[5]=1}
17    i:=i+1                              {i=6}
18  while i<=5 do                        {6<=5}
19    min:=a[1]                          {min=5}
20    max:=a[1]                          {max=5}
21    i:=2                                {i=2}
22    while i<=5 do                      {2<=5}
23      if min>a[i] then                  {5>4}
24        min:=a[i]                      {min=4}
25      if max<a[i] then                  {5<4}
26        i:=i+1                          {i=3}
27    while i<=5 do                      {3<=5}
28      if min>a[i] then                  {4>2}
29        min:=a[i]                      {min=2}
30      if max<a[i] then                  {5<2}
31        i:=i+1                          {i=4}
32    while i<=5 do                      {4<=5}
33      if min>a[i] then                  {2>3}
34      if max<a[i] then                  {5<3}
35      i:=i+1                            {i=5}
36    while i<=5 do                      {5<=5}
37      if min>a[i] then                  {2>1}
38        min:=a[i]                      {min=1}
39      if max<a[i] then                  {5<1}
40      i:=i+1                            {i=6}
41    while i<=5 do                      {6<=5}
42    writeln('MAX=',max)                 {writeln(5)}
43    writeln('MIN=',min)                 {writeln(1)}

```

图 8: 実行系列 2

2.5 関数呼び出し履歴を利用したプログラムスライス

2.5.1 関数呼び出し履歴を利用したプログラムスライスを計算するアルゴリズム

プログラムスライスの計算方法として [12] のアルゴリズムを用いる。このアルゴリズムは [8, 15] を参考にしてプログラム中の依存関係を調べ、それを元にプログラム依存グラフ (Program Dependence Graph 略して PDG) を作成する。

関数呼び出し履歴を利用したプログラムスライスを計算するために必要となる実行依存関係についての情報を、PDG を求める際に PDG の節点に与える。実行依存関係を以下のよに定義する。

1. 実行依存関係 (Execution Dependence, ED)

命令 s が実行された場合、命令 t も必ず実行されるとき、命令 t は命令 s に実行依存関係 $ED(s, t)$ がある。

2. 潜在的実行依存関係 (Potential Execution Dependence, PED)

命令 t が命令 s に潜在的実行依存しているとは、

- (a) 命令 s が実行されたとき命令 t も実行される可能性がある。
- (b) 命令 s が実行されなかったとき命令 t も実行されない。

ことを示して。命令 t が命令 s に潜在的実行依存していることを $PED(s, t)$ と表す。

上記の依存関係を元に以下に示す依存関係情報を PDG の各節点に与える。文 t が実行依存または潜在的実行依存している関数、手続き呼び出し文の集合を $CED(t)$ と書き、これを以下のように定義する。

$$CED(t) = \{ \text{関数, 手続き呼出文 } s \mid ED(s, t) \text{ または } PED(s, t) \}$$

関数呼び出し履歴を利用したプログラムスライスを計算するために、上記の依存関係とプログラムを実行して得られる動的な情報を用いる。動的な情報についての定義を以下に示す。

• 関数呼び出し履歴 (Calling History, CH)

入力を与えてプログラムを実行して、実行された関数呼び出し文の集合を関数呼び出し履歴と呼び CH と表す。

上記の依存関係情報を与えられたPDGを G , G に含まれる全ての辺の集合を E とすると, プログラム内の文 S の変数 v に関する関数呼び出し履歴を利用したプログラムスライスを表す節点の集合 V は以下のようにして計算される.

1. $V \leftarrow \{n | n \text{ は } S \text{ に対応する節点.}\}$
2. $N \leftarrow \{n | \langle v, n \rangle \in RD_{in}(S)\}$
3. $N \neq \phi$ の間以下の動作を繰り返す.
 - $l \in N$ を一つ選ぶ.
 - $N \leftarrow N - \{l\}$
 - $CED(l) \notin CH$ なら 3 の最初に戻る.
 - $V \leftarrow V \cup \{l\}$
 - $N \leftarrow N \cup \{k | (k \notin V) \wedge (k \xrightarrow{*} l \in E \vee k \dashrightarrow l) \wedge (k \text{ が } g\text{-in でない}) \wedge (k \text{ が } para\text{-in 節点でない})\}$ (ただし $k \xrightarrow{*} l$ は任意の変数の k から l への DD 関係辺を示す.)
 - k が $para\text{-in}$ 節点でも $para\text{-in}$ 節点でもないなら
 $N \leftarrow N \cup \{k | (k \notin V) \wedge (k \xrightarrow{*} l \in E \vee k \dashrightarrow l)\}$ (ただし $k \xrightarrow{*} l$ は任意の変数の k から l への DD 関係辺を示す.)

2.5.2 関数呼び出し履歴を利用したプログラムスライスの計算

上記の手順に従って 図9 のプログラムにたいして関数呼び出し履歴を利用したプログラムスライスを実際に計算する.

- 入力に $d = 0$ を与えて実行したときの, 図9 の 28 行目の文 `writeln(a)` に対応する PDG の節点の変数 a に関する関数呼び出し履歴を利用したプログラムスライスを 図 10 の右に示す.
- 入力に $d = 1$ を与えて実行したときの, 図9 の 28 行目の文 `writeln(a)` に対応する PDG の節点の変数 a に関する関数呼び出し履歴を利用したプログラムスライスを 図 10 の左に示す.

```

1  program double_or_tripple(input,output);
2  var a,b,c,d : integer;
3  procedure nijyou(var a,b : integer);
4  begin
5    a:=b*b
6  end;
7  procedure sanjyou(var a,b : integer);
8  begin
9    a:=b*b*b
10 end;
11 begin
12   b:=3;
13   c:=3;
14   readln(d);
15   if d=0 then
16     nijyou(a,b)
17   else
18     sanjyou(a,c);
19   writeln(a)
20 end.

```

図 9: プログラム

<pre> 1 program double_or_tripple(input,output); 2 var a,b,c,d : integer; 3 procedure nijyou(var a,b : integer); 4 begin 5 a:=b*b 6 end; 7 8 9 10 11 begin 12 b:=3; 13 14 readln(d); 15 if d=0 then 16 nijyou(a,b) 17 18 19 writeln(a) 20 end. </pre>	<pre> 1 program double_or_tripple(input,output); 2 var a,b,c,d : integer; 3 4 5 6 7 procedure sanjyou(var a,b : integer); 8 begin 9 a:=b*b*b 10 end; 11 begin 12 13 c:=3; 14 readln(d); 15 if d=0 then 16 17 18 sanjyou(a,c); 19 writeln(a) 20 end. </pre>
---	---

図 10: 異なる入力による関数呼び出し履歴を利用したプログラムスライス

3 プログラム文の依存関係を利用した開発システム

これまでに開発支援システムを作成した。このシステムはユーザーインターフェースにGUIを採用し、1つのウインド内でプログラムの編集、実行、デバッグが可能であり、またデバッグの際にプログラムスライスの利用が可能である。このシステムはプログラム文の依存関係を利用することによる開発作業の効率向上を目的とし開発した。本システムの記述言語として、プログラムの解析や実行などを行なう部分にはC言語を、インターフェース部分にはTcl/Tkを用いている。

3.1 システム構成

今回拡張したシステムは以下の6つの部分により構成されている。

- エディタ部

Pascalのサブセットで書かれたプログラムのロード、編集、セーブが行える。

- アナライズ部

プログラムの意味解析、依存解析を行なう。依存解析ではPDGを作成する。

- インタープリタ部

プログラムの実行を行なう。実行を行なう際には、実行系列を保存する。実行では、ステップ実行やブレークポイントの指定がおこなえる。

- 部分評価部

作成されたPDGを元にプログラムの部分評価を行なう。

- スタティックスライシング部

作成されたPDGを元にスタティックスライスを計算する。

- ダイナミックスライシング部

インタープリタ部により保存された実行系列を元にダイナミックスライスを計算する。

3.2 システム利用

ここでは、開発者(ユーザ)がプログラムを開発する際にどのような手順でシステムを利用するかを簡潔に述べる。まず、開発者(ユーザ)はコード化が終了したプログラムをインタプリタ部で実行する。そこで構文エラーが発見された場合は、プログラムをエディタ機能を用いて修正する。実行した結果実行した結果出力された値が誤っている場合には、ソースコード全体を対象にデバッグを行なうのではなく、そのソースコード(修正前プログラム)をスライサに入力し、誤った出力を行なった文についてのプログラムスライスのスライシング機能を用いて(スライス結果)を計算しそれを参照する、もしくはスライス結果をインタプリタ部でステップ実行しながら変数の値を参照することにより、エラーの位置を特定する。このとき、現在注目しているエラーに関係のない部分は参照しないので、比較的早くエラーの位置を特定できる。その後エディタ部でコードを修正し、修正した結果(修正済プログラム)をインタプリタ部で実行する。これら一連の作業を繰り返してデバッグを進めていく。

4 システムに拡張した機能

4.1 目的

入力のあるようなプログラムにおいては、スタティックスライスでは任意の入力が対象となっているため、エラーに関係のない実行されなかった文もデバッグ対象となる可能性がある。それに比べ、ダイナミックスライスは実行された文の集合である実行系列からスライスを計算するので、実行されなかった文がスライスに含まれるようなことはない。その結果、ダイナミックスライスの方がスタティックスライスより小さくなる。しかしダイナミックスライスは実行系列を保存しなければならないため、多くの時間と多くのメモリが必要であり実用的でない。関数呼び出し履歴を利用したプログラムスライスの機能を実現することで、スタティックスライスのコストで、ダイナミックスライスの結果に近いスライスが計算でき、さらに開発効率の向上が期待できる。本研究においては、関数呼び出し履歴を利用したプログラムスライスを抽出する機能を実現することにした。

4.2 操作法

関数呼び出し履歴を利用したプログラムスライスを求める際、本システムにおいては以下4つの操作が必要となる。

1. プログラムの入力.
2. プログラムの解析.
3. 関数の呼び出し履歴保存のためのプログラムの実行. 必要となる操作のうちこの操作のみが関数呼び出し履歴を利用したプログラムスライスの計算だけに必要な操作であるが、プログラムを実行すれば自動的に関数の呼び出し履歴が保存されるので特別な操作は必要ない.
4. スライシング基準の指定.

システムに表示されているプログラム内の文上でマウスの右ボタンを押すことで、その文がスライシング基準に指定され、図11のメニューが現れる。ここでどのタイプの

スライスを求めるかを指定する。指定すると 図 12 のメニューが現れる。このメニューでは、以下を指定する。

- **Varname**

指定された文で参照される変数の内その変数の関数呼び出し履歴を利用したプログラムスライスを求めるかを指定する。

- **Backward, Forward**

backward slice か forward slice を決定する。backward slice とは、スライスの起点となっている文に対して、影響を与えている文の集合である。forward slice とはスライスの起点となる文が、影響を与えている文の集合である。

- **Control flow**

制御依存関係を関数呼び出し履歴を利用したプログラムスライスの計算の対象とする。

- **Data flow**

データ依存関係を関数呼び出し履歴を利用したプログラムスライスの計算の対象とする。

- **Level**

PDGのノードを何段階まで辿るかを決定する。なにも設定しなければ、辿れるところまで辿る。



図 11: スライスの種類の選択



図 12: 変数の選択

4.3 内部仕様

ここでは内部仕様を簡単に述べる。関数呼び出し履歴を利用したプログラムスライスの計算は以下の3ステップに分かれる。

1. プログラムの解析を行ない,データ依存関係,制御依存関係を調べPDGを作成する.同時に実行依存関係も調べPDGの各節点に実行依存関係の情報を与える.
2. プログラムを実行し関数の呼び出し履歴を保存する. プログラムの実行はインタプリタにおいて行なわれる. インタプリタは構文木から文語ごとに実行する.
3. 指定されたスライス基準から関数呼び出し履歴を利用したプログラムスライスを計算する.

4.4 実行例

図13に関数呼び出し履歴を利用したプログラムスライスの実行例を示す. プログラムに入力を与え文 `writeln(a)` の変数 a についての関数呼び出し履歴を利用したプログラムスライスを計算している.

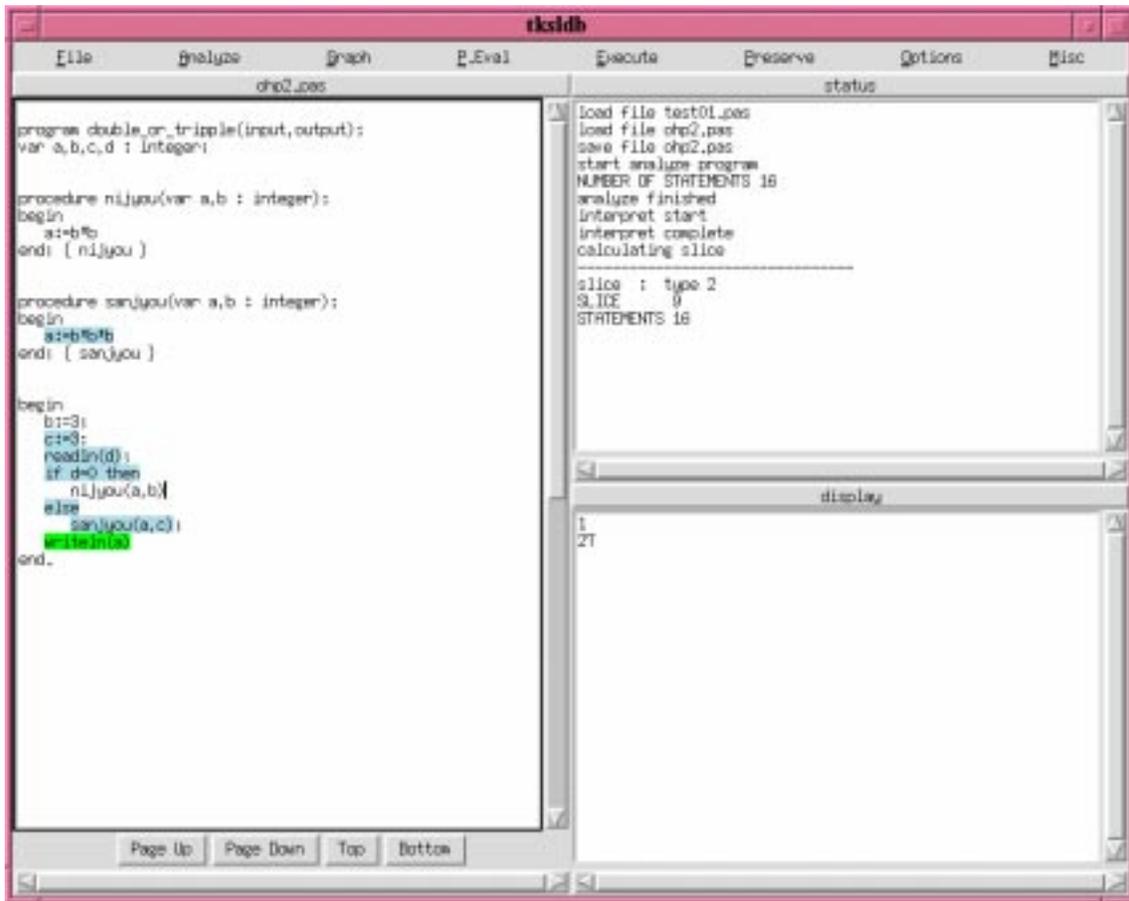


图 13: 実行画面

5 考察

本研究では、開発支援システムに関数呼び出し履歴を利用したプログラムスライスを計算する機能を追加した。関数呼び出し履歴を利用したプログラムスライスを抽出し、デバッグの対象として関数呼び出し履歴を利用したプログラムスライスの計算結果を参照することでデバッグ効率の向上を期待した。結果として以下のことがわかった。

- 入力データが与えられたとき、関数呼び出し履歴を利用したプログラムスライスを用いた場合の方が、スタティックスライスを用いた場合よりバグ位置特定が容易である。(表1)
- 関数呼び出し履歴を利用したプログラムスライスの計算は、スタティックスライスと同等のコストで行なえる。(表2, 表3)
- 関数呼び出し履歴を利用したプログラムスライスよりもダイナミックスライスの方が正確にバグ位置の特定が行なえる場合がある。(表2, 表3)

全ての可能な入力データにおけるプログラムの誤動作の原因となるバグの位置の特定にはスタティックスライスは有効である。しかし、ある特定の入力に対するプログラムの誤動作の原因となるバグの位置の特定には、関数呼び出し履歴を利用したプログラムスライスが有効である。このようにデバッグ時に2つのプログラムスライスを用いることでそれぞれの欠点を補い、デバッグ作業の効率向上が期待できる。

表 1: スライスのサイズの比較

プログラムサイズ スライスの種類	100 行	400 行
スタティックスライス	27 文	175 文
ダイナミックスライス	14 文	59 文
関数の呼び出し履歴を 利用したスライス	22 文	156 文

表 2: プログラムの解析時間の比較

P5-100MHz,memory 48MB での実行時間 (単位: ミリ秒)

プログラムサイズ スライスの種類	100 行	400 行
スタティックスライス	74	7603
関数の呼び出し履歴を 利用したスライス	76	7961

表 3: スライスの計算時間の比較

P5-100MHz,memory 48MB での実行時間 (単位: ミリ秒)

プログラムサイズ スライスの種類	100 行	400 行
スタティックスライス	5.4	83
ダイナミックスライス	800	13000
関数の呼び出し履歴を 利用したスライス	3.1	117

6 あとがき

本研究では、関数呼び出し履歴を利用したプログラムスライスを計算する機能を従来の開発支援システムに追加した。関数呼び出し履歴を利用したプログラムスライスを利用することでスタティックスライスより小さな文の集合に対して、デバッグ作業を限定して行なえる場合がある。またダイナミックスライスのように実行系列の保存なども必要なく、計算時間や実行系列を保存するためのメモリーなどが大量に必要となることもない。これによりスタティックスライスまたはダイナミックスライスを用いたデバッグ作業に比べ、作業時間の短縮が期待できる。

今後の課題としては、関数呼び出し履歴を利用したプログラムスライスの正確さをダイナミックスライスと同等にすること、関数呼び出し履歴を利用したプログラムスライスを計算する際に推定された実行パスの視覚化、システムの有効性の評価などがあげられる。

謝辞

本論文の作成において，常に適切な御指導および御助言を頂きました大阪大学 基礎工学部 情報工学科 井上 克郎 教授に深く感謝致します．

本論文の作成において，常に適切な御指導を頂きました同 楠本 真二 講師に深く感謝致します．

本論文を作成するにあたり，プログラムスライスに関する事項およびその他の面において多大な御指導，御助言を頂きました同 西松 顯 氏に深く感謝致します．

本論文を作成するにあたり，ツールの実装およびその他の面において多大な御指導，御助言を頂きました同 山本 哲男 氏に深く感謝致します．

最後に，その他の面で様々な御指導，御助言を頂いた大阪大学 基礎工学部 情報工学科 井上研究室の皆様に深く感謝致します．

参考文献

- [1] Aho, A.V., Sethi, S. and Ullman, J.D.:“Compilers : Principles, Techniques, and Tools”, Addison-Weseley, (1986).
- [2] Agrawal, H. and Horgan, J.,“Dynamic Program Slicing.” *SIGPLAN Notices*, Vol.25, No.6, pp.246-256(1990).
- [3] Agrawal, H., Demillo, R.A. and Spafford, E.H.:“Debugging with Dynamic Slicing and Backtracking”, *Software-Practice and Experience*, Vol.23(6), pp.589-616(1993).
- [4] Cordy, J.R., Eliot, N.L. and Robertson, M.G.:“TuringTool:A User Interface to Aid in the Software Maintenance Task”, *IEEE Transactions on Software Engineering*, Vol.16, No.3, pp.294-301(1990).
- [5] Gallagher, K.B. and Lyle, J.R.:“Using Program Slicing in Software Maintenance”, *IEEE Transactions on Software Engineering* , Vol.17, No.8, pp.751-761(1991).
- [6] Gupta, R. and Soffa, M.L.:“Hybrid Slicing : An Approach for Refining Static Slices Using Dynamic Information ”, *Proceedings of the third ACM Symposium on the Foundations of Software Engineering*, pp.29-40(1995).
- [7] Horwitz, S., Reps, T. and Binkley, D.:“Interprocedural Slicing Using Dependence Graphs”, *Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, pp. 35-46(1988).
- [8] Horwitz, S. and Reps, T.:“The Use of Program Dependence Graphs in Software Engineering”, *Proceedings of the 14th International Conference on Software Engineering*, pp.392-411(1992).
- [9] Jackson, D. and Rollins, E.J. :“A New Model of Program Dependences for Reverse Engineering”, *Proceedings of the Second ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp.2-10(1994).

- [10] Korel, b. and Laski, j., “Dynamic Program Slicing”. *Information Processing Letters*, Vol.29, No.10, pp.155-163(1988).
- [11] 西松 顕:“プログラム文の依存関係を利用した開発支援システムの機能拡張ダイナミックスライサの実現”,大阪大学基礎工学部情報工学科 特別研究報告 (1997-02).
- [12] 佐藤慎一, 飯田元, 井上克郎:“プログラムの依存関係解析に基づくデバッグ支援システムの試作”, 情報処理学会論文誌, vol.37, pp. 536-545(1996-4).
- [13] Shimomura, T.:“Bug Localization Based on Error-Cause-Chasing Methods”, Transactions of information Processing Society of Japan, Vol. 34, No. 3, pp. 489-500(1993).
- [14] 下村 隆夫:“プログラムスライシング技術と応用”, 共立出版 (1995).
- [15] 植田 良一, 練 林, 井上 克郎, 鳥居 宏次, “再帰を含むプログラムのスライス計算法”, 電子情報通信学会論文誌, vol. J78-D-I, pp. 11-22(1995).
- [16] Weiser, M.: “Program Slicing”, Proceedings of the Fifth International Conference on Software Engineering, pp. 439-449(1981).

付録

EBNF による対象言語の言語仕様

プログラム	= “program” プログラム名 “(” 名前の並び “)” “;” ブロック.
プログラム名	= 名前.
名前の並び	= 名前 {“,” 名前 }.
ブロック	= 変数宣言. 副プログラム宣言群 複合文.
変数宣言	= [“var” 変数宣言の並び].
変数宣言の並び	= 変数名の並び “:” 型 “;” { 変数名の並び “:” 型 “;” }.
変数名の並び	= 変数名 {“,” 変数名 }.
変数名	= 名前.
型	= 標準型 配列型.
標準型	= “integer” “boolean” “char”.
配列型	= “array” “[” 添字の最小値 “..” 添字の最大値 “]” “of” 標準型 .
添字の最小値	= 整数.
添字の最大値	= 整数.
整数	= [符号] 符号なし整数.
符号	= “+” “-”.
副プログラム宣言群	= { 副プログラム宣言 “;” }.
副プログラム宣言	= 副プログラム頭部 変数宣言 複合文 副プログラム頭部 “forward” “;”.
副プログラム頭部	= “function” 関数名 仮パラメータ “:” 標準型 “;” “procedure” 手続き名 仮パラメータ “;”.

関数名	= 名前.
手続き名	= 名前.
仮パラメータ	= [“(” 仮パラメータの並び “)”].
仮パラメータの並び	= [“var”] 仮パラメータ名の並び “:” 型 {“;” “var” 仮パラメータ名の並び “:” 型}.
仮パラメータ名の並び	= 仮パラメータ名 {“;” 仮パラメータ名 }.
仮パラメータ名	= 名前.
複合文	= “begin” 文の並び “end”.
文の並び	= 文 {“;” 文} [“;”].
文	= 基本文 “if” 式 “then” 限定文 “else” 文 “if” 式 “then” 文 “while” 式 “do” 文.
限定文	= 基本文 “if” 式 “then” 限定文 “else” 限定文 “while” 式 “do” 限定文.
基本文	= 代入文 手続き呼出文 入出力文 複合文 空文.
空文	= .
代入文	= 左辺 “:=” 式.
左辺	= 変数 関数名.
変数	= 純変数 添字付変数.
純変数	= 変数名.
添字付変数	= 変数名 “[” 添字 “]”.
添字	= 式.

手続き呼出文	= 手続き名 [“(” 式の並び “)”].
式の並び	= 式 {“,” 式}.
式	= 単純式 [関係演算子 単純式].
単純式	= [符号] 項 {加法演算子 項}.
項	= 因子 {乗法演算子 因子}.
因子	= 変数 定数 “(” 式 “)” 関数呼び出し “not” 因子.
関係演算子	= “=” “<>” “>” “>=” “<=” “<” .
加法演算子	= “+” “-” “or”.
乗法演算子	= “*” “/” “div” “mod” “and”.
関数呼び出し	= 関数名 [“(” 式の並び “)”].
入出力文	= “readln” [“(” 変数の並び “)”] “writeln” [“(” 出力指定の並び “)”].
変数の並び	= 変数 {“,” 変数}.
出力指定の並び	= 出力指定 {“,” 出力指定}.
出力指定	= 文字列 式.
定数	= 符号なし整数 文字列 “false” “true”.
符号なし整数	= 数字 {数字}.
文字列	= “” 文字列要素 {文字列要素} “”.
文字列要素	= アポストロフィ以外の任意の文字.
名前	= 英字 {英字 数字}.

英字 = "a" | "b" | "c" | "d" | "e" | "f" | ... | "z" |
"A" | "B" | "C" | "D" | "E" | "F" | ... | "Z" |
"_"

数字 = "0" | "1" | "2" | "3" | "4" |
"5" | "6" | "7" | "8" | "9".