特別研究報告

題目

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～

指導教官
井上克郎 教授

報告者
西松 顯

平成９年２月２１日

大阪大学 基礎工学部 情報工学科

平成10年度 特別研究報告

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～

西松 顯

内容梗概

プログラムスライス(Program Slice)には前方スライス,後方スライスの2種類ある．前方スライスとはプログラム中のある文sに現れる変数vの値に影響を与える全ての文の集合であり,後方スライスとはsに現れる変数wの定義が影響を与える全ての文の集合である．このプログラムスライスを用いることにより,デバッグ時にプログラム全体を扱うのではなく,誤った動作を行なう文に関係のある部分だけをプログラムから抽出し,その部分のみをデバッグの対象とすることでデバッグの作業効率の向上が期待できる．

プログラムスライスのうち、全ての可能な入力データに関するプログラム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(Execution Sequence)を解析して求められたプログラムスライスをダイナミックスライス(Dynamic Slice)と呼ぶ．ある入力に基づいたデバッグを行なう場合には,スタティックスライスよりもダイナミックスライスを用いた方が参照範囲を小さくすることができるので,バグ位置特定の時間が短縮できる．これまでにスタティックスライスを利用した開発支援システムを試作した．本研究では,開発支援システムにダイナミックスライサを実現する．これによりデバッグ効率の向上が期待できる．

主な用語

プログラムスライス(Program Slice)
ダイナミックスライス(Dynamic Slice)
スタティックスライス(Static Slice)
実行系列(Execution Sequence)

目 次

1 まえがき

現在の一般的なデバッグにおいてはプログラム全体を扱うのが原則である. しかし, 近年プログラムはより大規模で複雑なものになっている. このように大規模化, 複雑化したプログラムではデバッグ時にエラー原因の特定に時間がかかってしまい, 結果としてプログラム開発の効率が悪くなる.

このような場合に, エラーに関係があると思われる部分だけをプログラムから抽出するような機能があれば, 開発者はその部分だけに注目することができる. その結果, プログラム中のエラーの位置を発見する負担を軽減することができ開発効率の向上が期待される.

プログラムスライス (Program Slice)[10] 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の集合である. プログラムスライスのうち, 全ての可能な入力データに関するプログラム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)[2] と呼ぶ. このプログラムスライスを利用することにより、開発者がデバッグ時にエラーに関係のある部分のみを注目することができる。その結果、プログラム中のエラーの位置を発見する負担を軽減することができ, 開発効率の向上が期待される。

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. これらからスタティックスライスよりダイナミックスライスの方がスライスが小さくなる. よってダイナミックスライスを利用することで, 注目するべき文をさらに限定することができ, さらに開発効率の向上が期待できる.

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試作した. このシステムはスタティックスライスを利用し, デバッグを行なうことができる. 本研究では, このシステムにダイナミックスライスを抽出する機能を実現した.

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節でダイナミックスライサの説明を,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 [10] 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 [6] のアルゴリズムを用いる. このアルゴリズムは [5, 9] 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との間に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,[9]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. $V \leftarrow \{n | n$はS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.
- $N \leftarrow N - \{l\}$
- $V \leftarrow V \cup \{l\}$
- $N \leftarrow N \cup \{k | (k \notin V) \wedge (k \xrightarrow{\star} l \in E \vee k \dashrightarrow l) \wedge (k$が$g\text{–}in$でない$) \wedge (k$が$para\text{–}in$節点でない$)\}$(ただし$k \xrightarrow{\star} l$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
  $N \leftarrow N \cup \{k | (k \notin V) \wedge (k \xrightarrow{\star} l \in E \vee k \dashrightarrow l)\}$(ただし$k \xrightarrow{\star} l$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`l`を求めるための関数`lcm`や関数`lcm`を呼び出す文, 変数`l`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

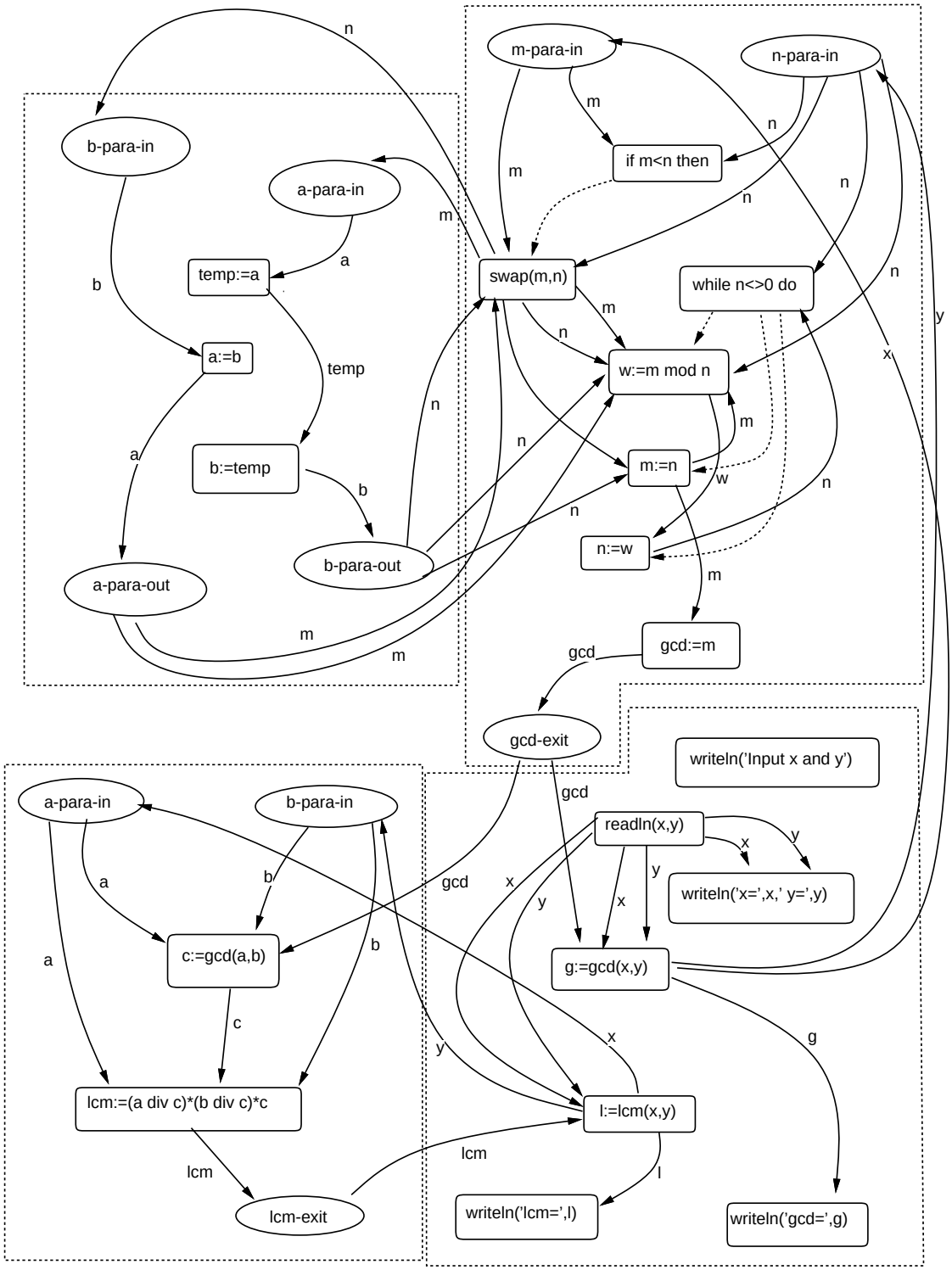
m-para-in
n-para-in
b-para-in
a-para-in
if m<n then
temp:=a
swap(m,n)
while n<>0 do
a:=b
w:=m mod n
b:=temp
m:=n
b-para-out
n:=w
a-para-out
gcd:=m
gcd-exit
writeln('Input x and y')
readln(x,y)
writeln('x=',x,' y=',y)
a-para-in
b-para-in
c:=gcd(a,b)
g:=gcd(x,y)
lcm:=(a div c)*(b div c)*c
l:=lcm(x,y)
lcm-exit
writeln('lcm=',l)
writeln('gcd=',g)

図 2: PDGの例

```
 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.
```

```
 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.
```

図 3: 元のプログラムとその Backward Slice の比較

```
 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.
```

```
 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.
```

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 =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 Ins(p) で表し, 実行時点 p に対して,Def(p) は実行時点 p における文 Ins(p) の実行により定義された変数,Use(p) は実行時点 p における文 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 DU(p, q) があるとは, ある変数 $w \in$ Use(q) が存在して,p =Def(q, w) の場合をいう. データ依存関係 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 $w \in$ 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,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 CtlExec(t) を次のように定義する.

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

   ただし, 命令 t がループ命令の場合には命令 t も 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 TC(p, q) があるとは,

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

   の場合をいう.

   制御依存関係 TC(p, q) は, 実行時点 q において命令 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$\mathrm{Ins}(p)$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$\mathrm{Ins}(q)$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$\mathrm{TC}(p,q)$があるという. このとき実行時点qにおける命令$\mathrm{Ins}(q)$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$VarREF(p)$と書き.これを以下のように定義する.

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

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$\mathrm{Control}(p)$と表す.

次にこれまでの手順により依存情関係報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い$C=(x,r,V)$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. $\mathrm{DS} \leftarrow \phi$

   $\mathrm{CalcDsObj} \leftarrow \phi$

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

3. $p \in$ CalcDsObj

   CalcDsObj $\leftarrow$ {実行時点q | $VarREF(p) \cup$ Contorol(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

```
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.
```

```
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.
```

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)}
```

図 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)}
```

図 8: 実行系列2

2.5 ダイナミックスライスとスタティックスライスの比較

ここで, これまでに述べたスタティックスライスとダイナミックスライスの比較を行なう. 具体的には, バグを含むプログラムでスタティックスライス, ダイナミックスライスを計算し２つのスライスの違いを見る.

2.5.1 ダイナミックスライスの長所

入力された数字の絶対値, 次乗を出力するプログラムを 図9 に示す. このプログラムは文12の`if x > 0 then`が`if x < 0 then`であるバグを含む. このプログラムに4を入力として与えたとき$|4| = -4(|4| = 4$が正しい$)$と出力される. 図10 , 図11 は文26の変数$a[2]$をスライシング基準としたときのプログラムスライスである. ダイナミックスライスでは, スライシング基準の指定の際, 配列の添字まで指定できる. しかし, スタティックスライスにおいては配列型変数は添字の値が決定できない場合があるので, 配列の添字まで指定できない. この例では, ダイナミックスライスは変数$a[2]$に関係ある部分のみ抽出されているが, スタティックスライスは変数aに関係のある部分が抽出されているため, 変数$a[2]$ に関係のない部分まで含まれている. このような場合, デバッグの際ダイナミックスライスを用いた方が効率が良い. このようにダイナミックスライスの方が参照する範囲が小さいためデバッグの効率が良いが, 入力を与え実行しないと計算できない.

```
 1 program pas104(output);
 2 var     n:integer;
 3         a:array [1..2] of integer;
 4
 5 function squere(x: integer): integer;
 6 begin
 7   squere := x * x
 8 end;
 9
10 function abs(x: integer): integer;
11 begin
12   if x < 0 then
13     abs := x
14   else
15     abs := -1 * x
16   end;
17
18 begin
19    a[1]:=0;
20    a[2]:=0;
21    writeln('INPUT 1 NUMBER');
22    readln(n);
23    a[1]:=squere(n);
24    a[2]:=abs(n);
25    writeln(n,'*',n,'=',a[1]);
26    writeln('|',n,'|','=',a[2]);
27 end.
```

図 9: プログラム

```
program pas104(output);
var     n:integer;
        a:array [1..2] of integer;

function squere(x: integer): integer;
begin
  squere := x * x
end;

function abs(x: integer): integer;
begin
  if x < 0 then
    abs := x
  else
    abs := -1 * x
  end;

begin
   a[1]:=0;
   a[2]:=0;
   writeln('INPUT 1 NUMBER');
   readln(n);
   a[1]:=squere(n);
   a[2]:=abs(n);
   writeln(n,'*',n,'=',a[1]);
   writeln('|',n,'|','=',a[2]);
end.
```

図 10: ダイナミックスライス

```
program pas104(output);
var     n:integer;
        a:array [1..2] of integer;

function squere(x: integer): integer;
begin
  squere := x * x
end;

function abs(x: integer): integer;
begin
  if x < 0 then
    abs := x
  else
    abs := -1 * x
  end;

begin
   a[1]:=0;
   a[2]:=0;
   writeln('INPUT 1 NUMBER');
   readln(n);
   a[1]:=squere(n);
   a[2]:=abs(n);
   writeln(n,'*',n,'=',a[1]);
   writeln('|',n,'|','=',a[2]);
end.
```

図 11: ダイナミックスライス

2.5.2 ダイナミックスライスの短所

図 12 のプログラムは入力された５つの数字の最大値, 最小値を出力するプログラムである. ただし文 15 が i<=5 でなく i>=5 になっているバグを含む.
入力に a[1]=4,a[2]=3,a[3]=5,a[4]=1,a[5]=2 を与えて実行すると出力は MAX=4,MIN=4 となる. 本来は MAX=5,MIN=1 と出力されるべきで, 変数 max,min の値が正しくない. そのうち変数 min の値のバグ原因を特定するために, 文 22 において変数 min についてスタティックスライスとダイナミックスライスを計算する. 図 13 がスタティックスライス, 図 13 がダイナミックスライスである. この例ではダイナミックスライスでは変数 min に間違った値を代入した文 12 は特定できるが, バグ位置である文 15 はダイナミックスライスに含まれていないのでバグ位置の特定が困難である. スタティックスライスについても, 実行されない文 18 が含まれるため変数 min が文 18 において値が定義されている可能性も考えてバグ位置特定を行なう必要がある. このような場合, ダイナミックスライスではバグ位置特定が困難である.

ただし, 利用者が本来この入力データではプログラムの文 18 が実行されることを知っている場合は, 次の手順により効率良くバグ位置の特定を行なえる.

1. ダイナミックスライスで文 18 は実行していないことが分かる.

   文 18 が実行されない原因として制御依存が考えられる.

2. スタティックスライスで文 18 の制御依存を調べる.

   文 17, 文 15 との制御依存関係を得る.

3. 文 17, 文 15 の分岐条件を調べる.

   文 15 の分岐条件では while 文内に入らない.

4. バグ位置である文 15 の特定.

このように両方を併用することでスタティック, ダイナミック単独で利用するより効率よくバグ位置特定をおこなえる.

```
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 12: プログラム

```
program max_min(input,output);
var i,max,min:integer;
    a:array[1..5] of integer;
begin
  i:=1;
  writeln('Input 5 numbers');
  while i<=5 do
  begin
    readln(a[i]);
    i:=i+1
  end;
  min:=a[1];
  max:=a[1];
  i:=2;
  while i>=5 do
  begin
    if min>a[i] then min:=a[i] ;
    if max<a[i] then max:=a[i] ;
    i:=i+1
  end;
  writeln('MAX=',max);
  writeln('MIN=',min)
end.
```

図 13: スタティックスライス

```
program max_min(input,output);
var i,max,min:integer;
    a:array[1..5] of integer;
begin
  i:=1;
  writeln('Input 5 numbers');
  while i<=5 do
  begin
    readln(a[i]);
    i:=i+1
  end;
  min:=a[1];
  max:=a[1];
  i:=2;
  while i>=5 do
  begin
    if min>a[i] then min:=a[i] ;
    if max<a[i] then max:=a[i] ;
    i:=i+1
  end;
  writeln('MAX=',max);
  writeln('MIN=',min)
end.
```

図 14: ダイナミックスライス

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タ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. 詳しい利用法については付録を参照されたい. 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ー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を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ー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) をインター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. これらから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ことが分かる. ダイナミックスライスを利用することで, エラーに関係のある文をさらに限定することができ, さらに開発効率の向上が期待できる. そこで, 本研究においては, ダイナミック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

ダイナミックスライサを使用する際, 本システムにおいては以下4つの操作が必要となる.

1. プログラムの読み込み.
2. プログラムの解析.
3. 実行系列の保存のためのプログラムの実行.
4. スライシング基準の指定.

上記の内3はダイナミックスライサのみに必要となる操作であり, また4についてはスタティックスライサの使用の際にも必要となるが, ダイナミックスライサの場合では異なるので, この２つについての操作法を述べる.

- **実行系列の保存**

実行系列は, 実行された文の集合であるのでプログラムを一度実行する必要がある. このとき, メニューバーの `OPTION` の `Save Ex_Sequence` をオンにして実行系列の保存を指定した後に実行する.

- **スライシング基準の指定**

ダイナミックスライシング基準は本来は, 実行系列上のある実行時点であるが, 今回作成したダイナミックスライサでは実行時点で指定するのではなくプログラム内の文でスライシング基準を指定することにした. while 文内にある文や, 関数, 手続き内にある文は実行系列上に同じ文が存在する可能性がある. この場合は実行系列でその文に該当するすべての実行時点をダイナミックスライシング基準としダイナミックスライスを計算し, 表示させることにした. 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, その文がダイナミックスライシング基準に指定され, 図 15 のメニューが現れる. ここでスタティックスライスかダイナミックスライスのどちらのスライスを求めるかを指定する. ここでは, ダイナミックスライスを指定する. すると 図 16 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ダイナミックスライスを求めるかを指定する. ここで変数 a の後についている [] は変数 a が配列であることを示している.

  - **Index Value**

    Varname において指定された変数が配列の場合は添字を指定することが可能である. 指定された添字の値を変数の添字がとらない場合はエラー出力を行なう. 添字を指定しない場合は, 配列がとりうる添字すべてが指定されたことになる.

  - **TC**

    制御依存関係のみをダイナミックスライスの計算の対象とする.

  - **DU**

    データ依存関係のみをダイナミックスライスの計算の対象とする.

図 15: スライスの種類の選択

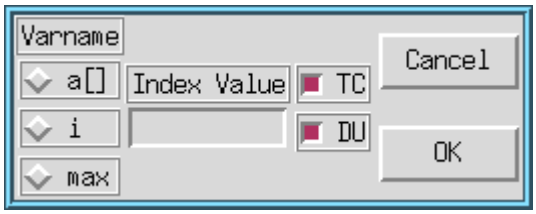

図 16: 変数の選択

4.3 内部仕様

ここでは内部仕様を簡単に述べる. ダイナミックスライサの実行は 2 ステップに分かれる. ステップ 1 では, 入力として構文解析, 意味解析の終了したプログラムの構文木, 変数情報をとり, これからプログラムの実行を行ない, 各実行時点間に依存関係を与える. 依存関係を与えられた実行時点は, ファイルに落す. これは, 実行系列が非常に長くなるとメインメモリでは保持しきれなくなるからである. プログラムの実行はインタープリタにおいて行なわれる. インタープリタは構文木から文ごとに実行する.

実行時点間の依存関係を求める手順は以下のとおりである.

- 実行している文で定義している変数が存在するなら, その変数を定義変数集合に登録する. 登録する情報としては, 変数名, 定義された実行時点番号, その変数の宣言された部分 (グローバルまたはローカルであるか, ローカルの場合は関数名, 手続き名を示す) がある.

- 実行している文で参照している変数が存在するなら, 定義変数集合から, その変数が定義された実行時点番号を得る. 実行時点番号をデータ依存関係情報として与える.

- 実行している文が, ある実行時点と制御依存関係があるとき制御依存関係がある実行時点番号を制御依存関係情報として与える.

ステップ 2 では, 指定されたスライシング基準からダイナミックスライスを計算する. スライシング基準の情報として, 文とその文で参照されている変数の 2 種類が与えられる. ファ

イルに落された実行系列から指定された文に合致する実行時点を検索し, その実行時点の依存関係情報からダイナミックスライスを計算する.

4.4 実行例

図 17 にダイナミックスライスの実行例を示す.
プログラムに入力を与え文 writeln('MIN',min) の変数 min についてのダイナミックスライスを計算している. 緑になっている文がスライシング基準で, 青がそのスライシング基準におけるダイナミックスライスである.

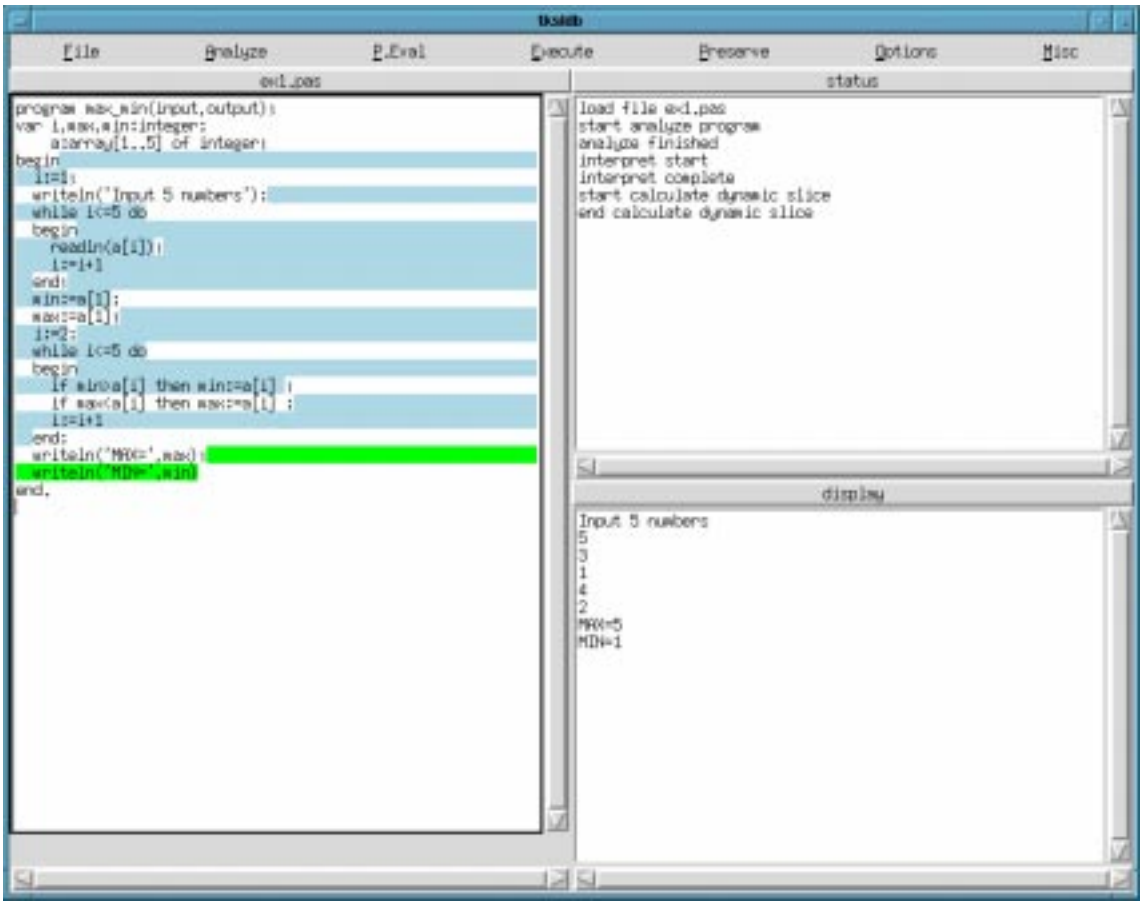

図 17: 実行画面

5　考察

本研究では,開発支援システムにダイナミックスライサを実現した. ダイナミックスライスを抽出し,デバッグの対象としてダイナミックスライスを参照することでデバッグ作業の効率の向上を期待した.結果として,以下のことが分かった.

1. 入力データが与えられた場合はダイナミックスライスを用いる方がスタティックスライスよりもバグ位置特定が容易である.

2. スタティックスライスではバグに配列型の変数が含まれる場合はバグ位置特定は難しいが,ダイナミックスライスを用いるとバグ位置特定が比較的容易である.

3. ダイナミックスライスではバグ位置特定が難しい場合がある.

全ての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特定はスタティックスライスを用いてデバッグを行なうと有効的である. 一方,特定の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特定はダイナミックスライスを用いてデバッグを行なうと有効的である. 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.

ただし実行系列に依存関係を与え保存する作業がある分ダイナミックスライスを利用する際,実行系列を保存しない場合よりも実行時間がかかる.

6　あとがき

本研究では, 開発支援システムにダイナミックスライサを実現した. ダイナミックスライスを利用することで, プログラムからバグに関係のある部分を抽出することができる. デバッグ時に開発者は抽出された部分をデバッグの対象とすることで, デバッグ作業時間の短縮が期待できる. ある入力におけるバグの位置特定を行なう場合, スタティックスライスではバグ位置特定が困難であるが, ダイナミックスライスは特定の入力データにおけるプログラムスライスであるので, プログラムの参照範囲が小さくなり, バグ位置特定が容易にできる.

今後の課題としてはダイナミックスライスを用いた機能 (例として, テストデータ自動生成機能) を実現すること, 他の言語への適用, システムの有効性の評価, スライス起点の指定方法の改善などがあげられる.

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 高田 智規 氏に深く感謝致します.

本論文を作成するにあたり，開発支援システムの開発/保守に関して御協力頂いた NTT データ通信株式会社 佐藤 慎一 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., Demillo, R.A. and Spafford, E.H.:“Debugging with Dynamic Slicing and Backtracking”, Software–Practice and Experience, Vol.23(6), pp. 589–616(1993).

[3] 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).

[4] 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).

[5] 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).

[6] 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, April 1996, pp. 536–545.

[7] 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).

[8] 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).

[9] 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, January 1995, pp. 11–22.

[10] Weiser, M.: “Program Slicing”, Proceedings of the Fifth International Conference on Software Engineering, pp. 439–449(1981).

付録

1. システムの言語仕様
2. システムのマニュアル

付録1 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”.