特別研究報告

題目

# プログラムスライスに基づく<br>デバッグ支援システムの実験的評価

指導教官
井上 克郎 教授

報告者
西江 圭介

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

# プログラムスライスに基づく<br>デバッグ支援システムの実験的評価

西江 圭介

## 内容梗概

ソフトウェア開発におけるテスト工程はテストとデバッグから構成されている．デバッグではフォールト位置の特定が最も時間を要する作業であると指摘されている．フォールト位置の特定を効率良く行う手法として，プログラムスライス技法が提案されている．プログラムスライスとは，プログラム中のある文の実行に影響を与える全ての文を抽出する技術である．これまでに我々の研究室ではプログラムスライスに基づくデバッグ支援システムを開発してきている．

本研究の目的は，プログラムスライス技法がフォールトの位置特定に有効であるかを実験的に評価することにある．具体的には，フォールトを含んでいるプログラムに対して被験者を 2 つのグループに分け，一方のグループの被験者は，デバッグ支援システムのスライス抽出機能を用いてデバッグを行い，もう一方のグループの被験者は，スライス抽出機能を用いずにデバッグを行う．そして 2 つのグループの間でデバッグに要した時間についての比較を行う．

実験の結果，システムのスライス抽出機能を用いてデバッグを行った方が，スライス機能を用いずにデバッグを行うよりも効率良くフォールトの位置特定ができており，スライシング技術の有効性が確認できた．

## 主な用語

プログラムスライシイング
デバッグ支援システム
実験的評価
平均値の差の検定

目 次

**1 まえがき** **4**

**2 プログラムスライシング** **6**

2.1 プログラム依存グラフ (PDG) · · · · · 6

2.2 プログラムスライス · · · · · 7

**3 デバッグ支援システム** **10**

3.1 言語仕様 · · · · · 10

3.2 機能 · · · · · 10

3.3 システムの使用方法 · · · · · 11

3.4 システム · · · · · 12

**4 評価実験** **14**

4.1 実験の概要 · · · · · 14

4.1.1 実験 1 · · · · · 14

4.1.2 実験 2 · · · · · 14

4.2 対象プログラムと仕様書 · · · · · 15

4.3 フォールトとテストデータ · · · · · 16

4.4 実験の設計，準備 · · · · · 18

4.5 手順 · · · · · 18

4.6 実験データ · · · · · 20

**5 分析** **22**

5.1 全体分析 · · · · · 22

5.1.1 実験 1 · · · · · 22

5.1.2 実験 2 · · · · · 23

5.2 各フォールトごとの分析 · · · · · 24

**6 考察** **25**

6.1 全体の考察 · · · · · 25

6.2 フォールトごとの考察 · · · · · 25

**7 あとがき** **27**

**謝辞** **28**

参考文献 **29**

付録 **30**

**1　スライシングシステムマニュアル** **31**

**2　酒屋問題仕様書** **38**

## 1　まえがき

ソフトウェアシステムの大規模化，複雑化にともないソフトウェア開発における生産性，及び，品質向上の実現はソフトウェア工学における研究の主要な目標に位置付けられてきている．ソフトウェアの品質や生産性を向上させるためには，開発されたソフトウェアプロダクトだけでなく，その開発プロセスを対象として作業の改善を行うことが必要である．

一方，現実のソフトウェアプロジェクトではソフトウェア開発コストの $50 \sim 80\%$ をテスト工程に費しているという報告がある．従って，ソフトウェア開発プロセスの改善を行うためには，テスト工程の改善を行うことが効果的である．テスト工程は故障の検出 (テスト) と故障の原因であるフォールトの修正 (デバッグ) の 2 つの作業から構成される．一般に，フォールトの位置を特定することがデバッグにおいて最も時間がかかると言われており [?, ?]，フォールトの位置を効率良く特定する方法の開発が重要視されている．

フォールトの位置を効率よく特定するための方法の一つとして，プログラムスライス技法 (Program Slicing) を利用した手法が提案されてきている [?]．プログラムスライス技法はプログラム内のある文の実行に影響を与える全ての文を抽出する技法であり，抽出された文の集合をスライスと呼ぶ．プログラムスライス技法を利用するデバッグでは値の誤っている変数に対して，全プログラムにわたってスライスを求め，そのスライスの中でフォールトとなっている文を捜し出す．

これまでに，文献 [8] において，静的スライス (プログラムの依存関係を解析し，それに基づいて求められるスライス) を用いたデバッグ支援システムを開発してきている．このシステムを用いることにより，プログラム中でフォールトに関係のある部分のみを作業の対象とすることができるため，効率の良いデバッグを行うことが期待される．

しかし，実際にプログラムスライス技法がデバッグにおけるフォールトの位置特定に有効であるかは評価されていない．そこで本研究では プログラムスライス技法がフォールトの位置特定に有効であるかを実験的に評価する．具体的には，フォールトを含んでいるプログラムに対して被験者を 2 つのグループ (G1，G2) に分け，あらかじめフォールトを含んでいるプログラムに対して G1 に含まれている被験者は，デバッグ支援システムのスライス抽出機能を用いてデバッグを行い，G2 に含まれている被験者は，スライス抽出機能を用いずにデバッグを行う．そして G1，G2 の間でデバッグに要した時間についての比較を行う (実験 1)．さらに，新たにフォールトを含んでいる別のプログラムを用意し，G1 に含まれている被験者にはスライス抽出機能を用いずにデバッグを行い，G2 に含まれている被験者は，スライス抽出機能を用いてデバッグを行う．そして G1，G2 の間でデバッグに要した時間についての比較を行う (実験 2)．最後に，実験 1，実験 2 のそれぞれについてスライス機能を使用したグループと，使用しなかったグループのフォールト位置特定時間を平均値の差の検定に

より評価する [?] .

以降 , 2 . ではデバッグ支援システムの基礎となっているプログラムスライシングについて述べる . 3 . では , デバッグ支援システムの機能について紹介する . 4 . では実際に行った評価実験について述べる . 5 . では評価実験で得られた結果について分析を行う . 6 . では分析結果の考察を行う . 最後に 7 . でまとめと今後の課題を記す .

## 2　プログラムスライシング

本実験で使用するシステムは，プログラムの構文解析を行い，その後プログラムの依存関係の解析を行う．依存解析の結果，プログラム依存グラフ (Program Dependence Graph，PDG) を得る．このPDGを用いてスライシングを行うことによりプログラムの参照範囲を小さくし，効率的なデバッグの支援を行う．

以下にプログラムスライシングについて述べる．

### 2.1　プログラム依存グラフ (PDG)

PDGはプログラム内の文の変数の依存関係を表すグラフである．[?, ?, ?]プログラム中の各文に対応した頂点 (節点とも呼ぶ) と，それらを結ぶ辺から構成されている．辺には変数の定義，参照関係を伝えるデータ依存辺と，条件式とその結果によって実行が影響を受ける文の関係を伝える制御依存辺がある．

データ依存 (Data Dependence，DD) は，PDGの各頂点の到達定義集合 (Reaching Definition，RD) を求めることにより得られる．ある頂点tのRDは変数vと頂点sの組 (v，s) の集合であり，以下の2つのことを示している．

1. プログラム中の文sで変数vを定義している．
2. プログラム中の2つの文s1，s2への全ての実行パス中に，vを定義しないパスが少なくとも1つは存在する．

s2のRDに (v，s1) が含まれ，かつs2がvを参照するとき，s1からs2へのDD関係があるという．

制御依存 (Control Dependence，CD) は，ある条件判定部分s1の結果により文s2の実行の有無が決まるときのs1，s2の間の依存関係を表す．

一般のプログラムには，複数の手続きが定義されており，各手続き間で引数や大域変数を通じてDD関係が生じる．これらのDD関係を表すためにプログラム中の文とは直接対応しない頂点 (特殊節点) を用意する．

PDGの作成は以下のように行われる．

1. プログラムを解析する
2. プログラムの各文を頂点にきりわける
3. それぞれの頂点におけるRDを求める
4. 3. をもとにCD辺，DD辺を生成する

例として図1のプログラムに対応するPDGを図2に示す．

### 2.2 プログラムスライス

文sにおける変数vに関するスライスとは，PDG上においてデータ依存辺または制御依存辺を辿って文sの変数vに到達できる頂点の集合に対応する文の集合である．[?]また文sをスライシング基準(slicing criterion)と呼び，PDGをスライシング基準から辺を順方向に探索して得られるスライスをforward sliceと呼び，逆方向に探索して得られるスライスをbackward sliceと呼ぶ．

プログラムスライスとは，プログラム中のある着目したスライシング基準から実行順方向(forward)または，実行逆順方向(backward)にスライシングを行った結果として得られる文の集合のことを言う．

```
 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: プログラム例

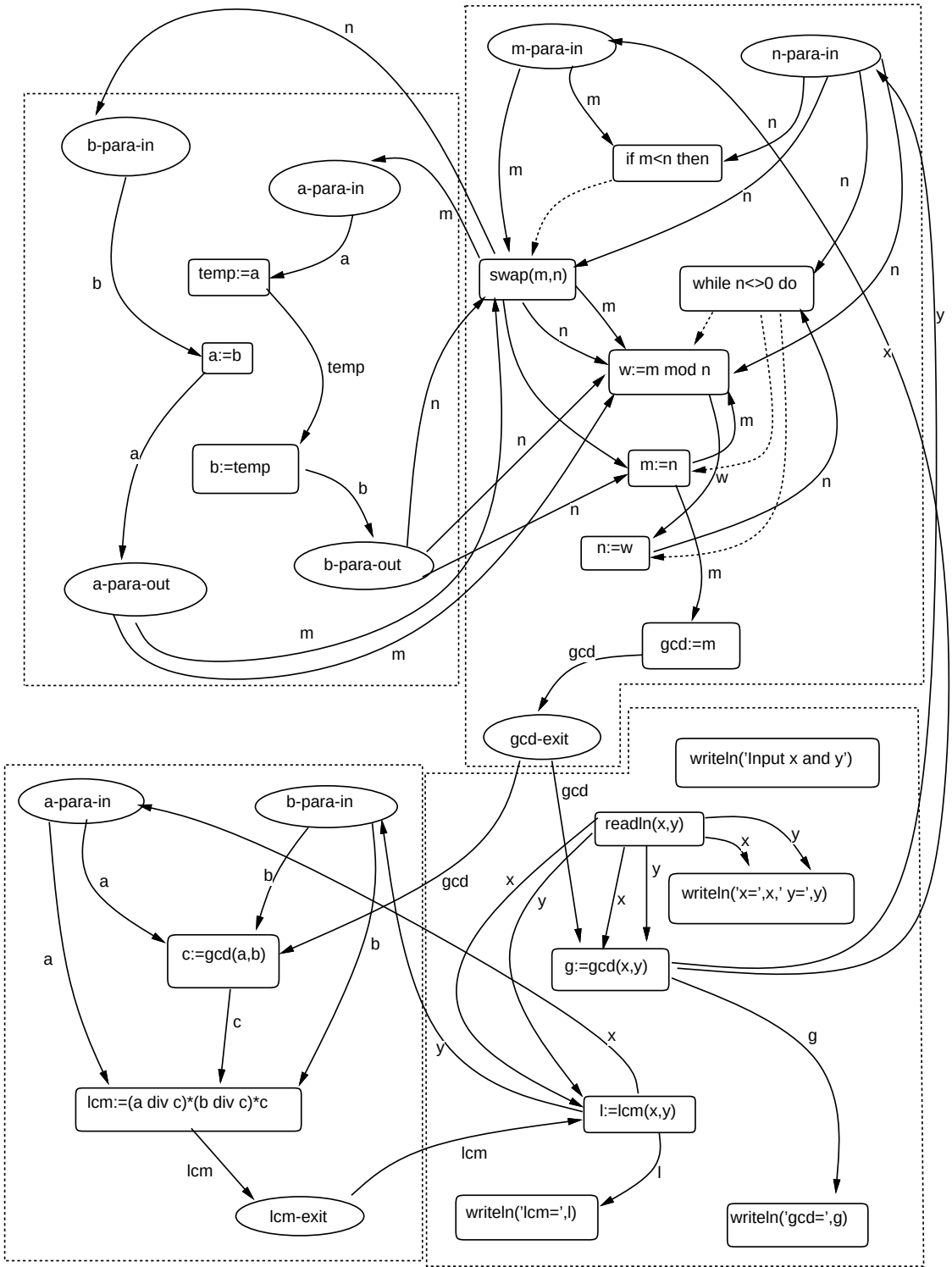



図 2: 図 1 のプログラムの PDG の

## 3 デバッグ支援システム

本実験で使用する，プログラムスライスに基づくデバッグ支援システム (以下，本システムと呼ぶ) は実装 (解析部，実行部) には C 言語，インタフェースには Tcl/Tk を用いた．

### 3.1 言語仕様

本システムが対象とする言語は，以下のような仕様を持つ Pascal サブセットである．

- 文として，代入文，条件文 (if 文)，繰り返し文 (while 文)，手続き呼び出し文，begin-end で囲まれる複合文を扱う．
- 手続きは自己再帰，及び，相互再帰呼び出しが可能である．
- 変数のデータ型はスカラー型のみでポインタ型は扱わない．具体的には整数型，文字型，論理型およびそれらを要素に持つ配列型とする．

このように，簡単化されてはいるが，プログラミング言語の基本的な文の構造は全て含んでいるため，拡張は容易に行うことができる．

### 3.2 機能

本システムの機能は，

f1 トレース機能
通常のデバッガと同様にプログラムの連続実行，ステップ実行，変数の参照，変数の設定，ブレークポイントの設定，などの機能．

f2 スライス機能
プログラム中のある文のある変数を指定すると，そこをスライシング基準としたスライスが抽出できる．このスライスは forward slice，backward slice のどちらでも抽出できる．またそのときに，PDG の制御依存辺とデータ依存辺の片方だけを探索したり，希望の段階まで探索することもできる．

f3 スライス部分実行機能
f2 で抽出されたスライス部分を 1 つのプログラムとして，そのスライス部分のみ実行させる機能．

である．

### 3.3 システムの使用方法

本システムを用いてデバッグを行うには以下の 5 つの手順がある．

1 プログラムの入力
デバッグ対象のプログラムを読み込む

2 プログラムの解析
プログラムの構文解析と依存解析を行う．

3 実行 (機能 f1)
プログラムを実行させる．(通常のデバッガの機能)

4 スライシング (機能 f2)
スライスを抽出し，その結果を表示する．

5 スライス部分実行 (機能 f3)
スライシングを行った場合，スライス部分のみを実行させることができる．

具体的には 1， 2，を行った後に 3，4，5 のステップを随時用いたり，併用することでデバッグを行う (図 3)．なお，詳細な使用方法は，付録のスライシングシステムマニュアルを参照されたい．

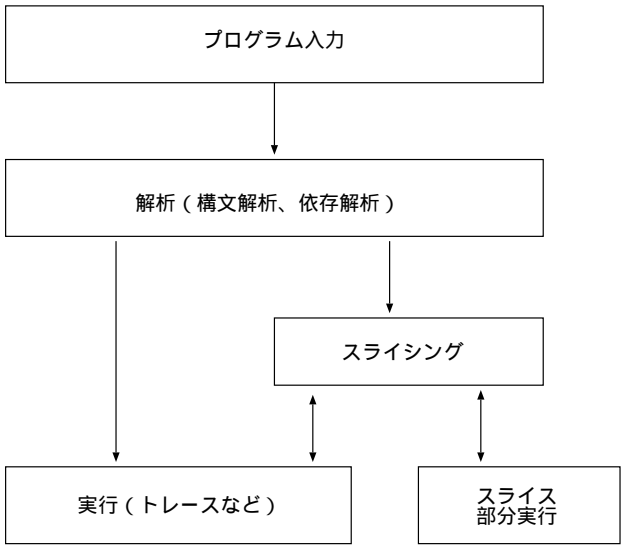



図 3: システム使用方法概略

### 3.4 システム

次ページの図4は，本システムによりスライシングを行った結果である．スライシング基準は濃い網掛けで記された文の変数xであり，ここからデータ依存関係と，制御依存関係の両方についてbackward sliceを計算した結果である．(スライス結果は，薄い網掛け部分である)もし変数xにフォールトの症状が見られた場合，このスライス部分に原因があることになる．

```
program samp(input,output);
var x,y,ans : integer;

procedure kai(var int:integer);
var temp,z:integer;
begin
   z:=0;temp:=int;
   while temp>=1 do
   begin
      z:=temp*z;
      temp:=temp-1
   end;
   int:=z
end; { kai }


procedure power(var int :integer);
var temp,z : integer;
begin
   z:=1;temp:=int;
   while temp>=1 do
   begin
      z:=int*z;
      temp:=temp-1
   end;
   int:=z
end; { power }


begin
   readln(x);
   y:=x;
   kai(x);
   power(y);
   if x>y then
      ans:=x-y
         else
            ans:=y-x;
   writeln(ans)
end.
```

図 4: システム

# 4　評価実験

本実験の目的はプログラムスライシングが，フォールトの位置特定に有効であるか調べることである．本実験全体は 2 回の実験から構成されている．(実験 1，実験 2)

## 4.1　実験の概要

### 4.1.1　実験 1

実験の被験者 6 人を 2 つのグループ G1，G2 に分ける．G1 に含まれる被験者 (A1，A2，A3) は，システム上の f1 ~ f3 までの全ての機能を使用できる．つまりスライス機能とトレース機能の両方を用いてフォールトの位置特定を行ってもらう．G2 に含まれる被験者 (B1，B2，B3) は，システムの f1 の機能しか使用できない．つまり G2 の被験者は通常のデバッガのもつトレース機能しか使用できない．

### 4.1.2　実験 2

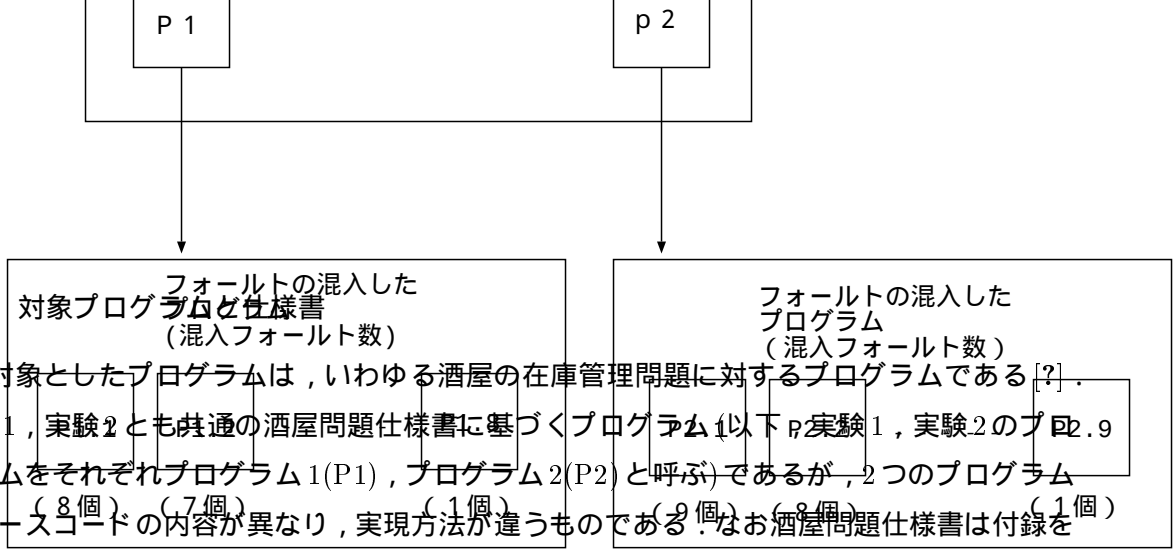
実験 1 で G1 に含まれていた被験者は，実験 2 ではシステム上の f1 の機能しか使わせない．つまり実験 1 で G2 の被験者が使ったトレース機能しか使わずにフォールトの位置特定を行ってもらう．逆に実験 1 で G2 に含まれていた被験者は，システム上の f1 ~ f3 の機能を全てを使用できる．

表 1: 被験者のグループ分け (実験 1)

| グループ | 人数 | 使用可能なシステムの機能 |
|---|---|---|
| G1 | 3 人 (A1 ~ A3) | スライス機能, トレース機能 (f1 f2 f3) |
| G2 | 3 人 (B1 ~ B3) | トレース機能 (f1) |

表 2: 被験者のグループ分け (実験 2)

| グループ | 人数 | 使用可能なシステムの機能 |
|---|---|---|
| G1 | 3 人 (A1 ~ A3) | トレース機能 (f1) |
| G2 | 3 人 (B1 ~ B3) | スライス機能, トレース機能 (f1 f2 f3) |




### 4.2 対象プログラムと仕様書

対象としたプログラムは，いわゆる酒屋の在庫管理問題に対するプログラムである[?]．実験1，実験2とも共通の酒屋問題仕様書に基づくプログラム(以下，実験1，実験2のプログラムをそれぞれプログラム1(P1)，プログラム2(P2)と呼ぶ)であるが，2つのプログラムはソースコードの内容が異なり，実現方法が違うものである．なお酒屋問題仕様書は付録を参照されたい．

実験者は，P1について8個，P2について9個のバージョンを用意する(図5)．

図 5: 対象プログラムの構成

### 4.3 フォールトとテストデータ

プログラムにあらかじめ入れておいた，フォールトの個数は，プログラム 1 が 8 個 (F1.1 ~ F1.8)，プログラム 2 が 9 個 (F2.1 ~ F2.9) である．それぞれのフォールトはそれに対応したテストデータを持っている．つまりあるフォールトはそれと対応したテストデータでないと発見できない．それぞれのフォールトの内容は以下のようになっている (表 3，表 4)．

表 3: プログラム 1 中のフォールト

| フォールト番号 | 内容 | 発見できるテストデータ |
|---|---|---|
| F1.1 | 出力処理の不足 | T1.1 |
| F1.2 | 変数の誤り | T1.2 |
| F1.3 | 条件文の誤り | T1.3 |
| F1.4 | 配列の初期化洩れ | T1.4 |
| F1.5 | 手続きの未実現 | T1.5 |
| F1.6 | データ更新の誤り | T1.6 |
| F1.7 | 手続きのパラメータ渡しの誤り | T1.7 |
| F1.8 | 手続きの実行位置の誤り | T1.8 |

表 4: プログラム 2 中のフォールト

| フォールト番号 | 内容 | 発見できるテストデータ |
|---|---|---|
| F2.1 | 条件文の誤り | T2.1 |
| F2.2 | 条件文の誤り | T2.2 |
| F2.3 | 添字ミス | T2.3 |
| F2.4 | 手続きの実行順序の誤り | T2.4 |
| F2.5 | 手続きのパラメータ渡しの誤り | T2.5 |
| F2.6 | 手続きの未実現 | T2.6 |
| F2.7 | データ更新の誤り | T2.7 |
| F2.8 | 変数の誤り | T2.8 |
| F2.9 | 変数の誤り | T2.9 |

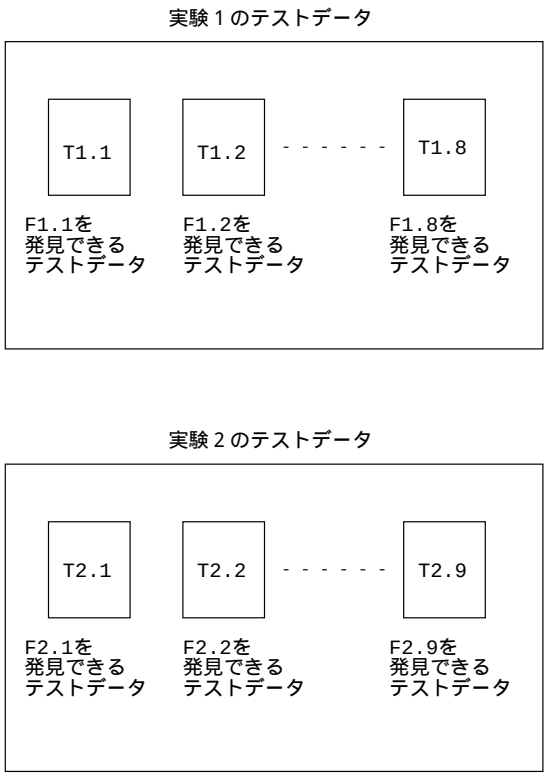



図 6: テストデータとフォールトの関係

Fi.j,Ti.j はそれぞれ,i が実験 (1：実験 1,2：実験 2),j がフォールトの番号を表す.

### 4.4 実験の設計，準備

実験 1 を開始する前に，被験者全員に本実験の対象プログラムである酒屋問題の仕様書を配布し，全員に説明を行った．また，全員に対し本システムの使い方についての講義を行った．その後，実験 1 でスライスを用いてデバッグを行う被験者グループ G1 に対して，スライスに関する講義を行った．更に，実験 2 を開始する前に，実験 2 でスライスを用いてデバッグを行う被験者グループ G2 に対して，スライスに関する講義を行った．

### 4.5 手順

実験は被験者一人づつを別室に隔離して行い，被験者がフォールトの位置特定ができれば実験監督者 (実験者) にフォールトの位置を報告するようにした．以下，実験の手順について記すが，実験 1，実験 2 とも同様の手順で行っているので，実験 1 のみ記す． 下記の表 (表 5) は，プログラムのバージョンとテストデータの対応表である．

表 5: プログラム 1(P1) のバージョンとテストデータの対応表

| バージョン | 表記 | 含んでいるフォールト | 対応するテストデータ |
|---|---|---|---|
| 1 | P1.1 | F1.1 ~ F1.8 | T1.1 |
| 2 | P1.2 | F1.2 ~ F1.8 | T1.2 |
| 3 | P1.3 | F1.3 ~ F1.8 | T1.3 |
| 4 | P1.4 | F1.4 ~ F1.8 | T1.4 |
| 5 | P1.5 | F1.5 ~ F1.8 | T1.5 |
| 6 | P1.6 | F1.6 ~ F1.8 | T1.6 |
| 7 | P1.7 | F1.7 ~ F1.8 | T1.7 |
| 8 | P1.8 | F1.8 | T1.8 |

F1.1を発見できる
テストデータ

F1.2を発見できる
テストデータ

F1.8を発見できる
テストデータ

STEP1 ：j:=1 とする．

STEP2 ：テストデータ T1.j を用いて，プログラム P1.j のフォールト (F1.j) の位置特定を行う．

STEP3 ：特定したフォールトとその位置を実験監督者 (実験者) に申告する．正しい場合は STEP4 へ，間違った場合は STEP2 へ戻る．

STEP4 ：i==8 の場合実験終了．i < 8 の場合，i:=i+1 として STEP2 へ戻る．

図 7: 実験の手順

### 4.6 実験データ

実験で得られたデータは

1 . 各被験者が各フォールトの位置特定に要した時間 (単位：分)

2 . 各被験者が全てのフォールトの位置特定に要した時間 (単位：分)

3 . スライシング基準の取り方 (スライス機能を用いた被験者)

である．

1. については，フォールト認識からフォールト位置特定までの時間を計測している．ここでフォールト認識とは，与えられたテストデータを実行することにより誤った出力結果であることを認知することである．しかし，ほとんどのテストデータは実行させればフォールトの症状がすぐに認知できるものであったので，実際にはテストデータを被験者に与えた時間からフォールト位置特定までの時間が，被験者のフォールト位置特定に要した時間とほぼ同じである．2. については 1 で得られた各フォールトの位置特定に要した時間を実験 1，実験 2 それぞれについて合計したものである (1. と 2. の結果は表 6，表 7 参照)．3. については，スライシングを用いた被験者のみについて得られるデータである．

表 6: 実験 1

| G1(スライス使用) | F1.1 | F1.2 | F1.3 | F1.4 | F1.5 | F1.6 | F1.7 | F1.8 | 計 |
|---|---|---|---|---|---|---|---|---|---|
| A1 | 31 | 8 | 15 | 25 | 14 | 15 | 4 | 7 | 119 |
| A2 | 26 | 10 | 15 | 20 | 26 | 10 | 12 | 9 | 128 |
| A3 | 17 | 20 | 13 | 22 | 18 | 10 | 8 | 12 | 120 |
| G2(スライス不使用) | F1.1 | F1.2 | F1.3 | F1.4 | F1.5 | F1.6 | F1.7 | F1.8 | 計 |
| B1 | 17 | 10 | 26 | 27 | 35 | 17 | 7 | 15 | 154 |
| B2 | 28 | 18 | 36 | 17 | 25 | 23 | 16 | 12 | 175 |
| B3 | 23 | 12 | 28 | 32 | 41 | 17 | 7 | 6 | 166 |

表 7: 実験 2

| G2(スライス使用) | F2.1 | F2.2 | F2.3 | F2.4 | F2.5 | F2.6 | F2.7 | F2.8 | F2.9 | 計 |
|---|---|---|---|---|---|---|---|---|---|---|
| B1 | 18 | 6 | 27 | 18 | 20 | 20 | 8 | 2 | 12 | 131 |
| B2 | 11 | 8 | 10 | 16 | 16 | 13 | 7 | 7 | 5 | 92 |
| B3 | 14 | 14 | 19 | 36 | 10 | 5 | 17 | 1 | 2 | 118 |
| G1(スライス不使用) | F2.1 | F2.2 | F2.3 | F2.4 | F2.5 | F2.6 | F2.7 | F2.8 | F2.9 | 計 |
| A1 | 17 | 6 | 12 | 30 | 6 | 11 | 5 | 5 | 26 | 118 |
| A2 | 17 | 5 | 24 | 13 | 18 | 16 | 19 | 5 | 9 | 126 |
| A3 | 36 | 24 | 12 | 41 | 8 | 15 | 5 | 4 | 10 | 155 |

表中の Fi.j はフォールトの番号，A1 ~ A3，B1 ~ B3 は被験者，数字はフォールトの位置特定にかかった時間 (単位：分)

# 5　分析

実験により得られたデータを

1 全体分析

2 各フォールトごとの分析

の 2 種類のアプローチから分析を行う．

1 の全体分析とは，各被験者の，デバッグ開始 (フォールト認識) から終了 (フォールト位置特定) までの総時間を，G1，G2 のグループの平均に対して平均値の差の検定 [?] を行い，スライス機能を使用した被験者が実際に効率良くフォールトの位置特定を行っているか分析をする．2 の各フォールトごとの分析とは各被験者のフォールトごとの位置特定時間の平均を，G1，G2 のグループの平均に対して平均値の差の検定を 1 と同様に行い，特にスライス機能を使用した被験者が効率良くフォールト位置特定を行っているフォールトが存在するか，または逆の場合が (トレース機能のみを使用した被験者の方が効率良くフォールト位置特定を行っている) 存在するかを分析している．

## 5.1　全体分析

### 5.1.1　実験 1

G1(スライスを使用したグループ) の被験者 (A1 ~ A3) の，デバッグ開始 (フォールト認識) から終了 (フォールト位置特定) までの総時間はそれぞれ 119 分 (A1)，128 分 (A2)，120 分 (A3) であるので，平均は 122 分である．

G2(スライスを使用していないグループ) の被験者 (B1 ~ B3) の，デバッグ開始 (フォールト認識) から終了 (フォールト位置特定) までの総時間はそれぞれ 154 分 (B1)，175 分 (B2)，166 分 (B3) であるので，平均は 155 分である．

上記の結果をウェルチの検定で行ったところ，スライスを用いた方が短時間で全てのフォールトを発見していることがわかる (有意水準 1%)．

### 5.1.2 実験 2

G2(スライスを使用したグループ)の被験者(B1 ~ B3)の，デバッグ開始(フォールト認識)から終了(フォールト位置特定)までの総時間はそれぞれ 118 分(B1)，126 分(B2)，155 分(B3)であるので，平均は 114 分である．

G1(スライスを使用していないグループ)の被験者(A1 ~ A3)の，デバッグ開始(フォールト認識)から終了(フォールト位置特定)までの総時間はそれぞれ 131 分(A1)，92 分(A2)，118 分(A3)であるので，平均は 133 分である．

上記の結果より，スライスを用いた方が短時間で全てのフォールトを発見していることがわかる．しかし，実験 1 と同様にウェルチの検定で行ったところ有意差は，実験 1 ほど顕著ではなかった(約 15%の有意水準)．

### 5.2 各フォールトごとの分析

実験 1，実験 2 ともにスライスを用いた方が，非常に短時間での位置特定ができているフォールトが存在した．これらのフォールトはスライスを使用したグループとスライスを使用してないグループで平均値の差の検定を行ったところ有意差の見られるものであった (有意水準 5%以下)．

しかし，スライスを用いない方が明らかに短時間で位置特定ができているフォールトは，実験 1，実験 2 を通しても存在しなかった．

表 8: フォールトごとの平均値

| 実験 1 | F1.1 | F1.2 | F1.3 | F1.4 | F1.5 | F1.6 | F1.7 | F1.8 |
|---|---|---|---|---|---|---|---|---|
| G1(スライス使用) | 25 | 13 | 14 | 22 | 19 | 12 | 8 | 9 |
| G2(スライス未使用) | 23 | 13 | 33 | 25 | 34 | 19 | 10 | 11 |

| 実験 2 | F1.2 | F2.2 | F2.3 | F2.4 | F2.5 | F2.6 | F2.7 | F2.8 | F2.9 |
|---|---|---|---|---|---|---|---|---|---|
| G2(スライス使用) | 13 | 9 | 19 | 23 | 15 | 13 | 11 | 3 | 6 |
| G2(スライス未使用) | 23 | 12 | 16 | 28 | 11 | 14 | 10 | 5 | 15 |

表 9: 実験 1, 実験 2 で特に有意な差が見られたフォールト例

| フォールト番号 | 内容 |
|---|---|
| F1.3 | 条件文の誤り |
| F1.5 | 手続きの未実現 |
| F1.6 | データ更新の誤り |
| F2.1 | 条件文の誤り |
| F2.9 | 変数の誤り |

# 6 考察

## 6.1 全体の考察

実験2において，スライスを使用したグループの方が短時間でフォールトの位置特定を行っているにもかかわらず，統計的に有意差が見られなかった．これは実験の被験者の数が6人(3人+3人)と少なかったためである．例えば，実験2の4番目のフォールト(F2．4)などはスライスを使用したグループ(G2)の平均位置特定時間が23分であるのに対して，使用していないグループ(G1)は28分であったにもかかわらず，有意差が現れていない．これはG2の被験者B3がこのフォールトの位置特定に36分要していることが(他の2人は18分，16分)大きく統計的(分散)に影響を与え，平均値の上では有意差として現れてこないためである．しかし，逆にスライスを用いた方が，用いない方よりもフォールト位置特定時間がかかったという例は見られなかった．

またスライス部分実行機能(f3)は実際に抽出したスライス部分のみを実行させる機能で，着目した変数に無関係な部分は全く実行されない．この機能はプログラムの参照範囲を小さくしフォールト位置特定に有効であることが期待されたが，実際のデバッグ作業において被験者はこの機能を使用しなかった．これは通常スライスには出力文(writeln文)が含まれないため出力結果が参照できず，使いにくいことが実験後の被験者へのインタビューでわかっている．このことよりf3機能のユーザーインタフェースの改善を行うことでスライス機能の有効性の更なる向上が期待できる．

## 6.2 フォールトごとの考察

本実験の中でスライスを使用すれば明らかに位置特定ができるものが幾つか存在している．これらのフォールトには次のような特徴が見られた．

1．フォールト原因である変数の特定が容易

フォールト原因である変数がわかれば，そこを基準にしてスライスを抽出することにより，簡単にプログラムの参照範囲を小さくすることができるため

2．類似した手続き，ルーチンが存在する

類似した手続き，ルーチンが存在すると，スライスを抽出すればその部分が顕著に現れ，比較が容易になる

3．プログラム中に広範囲にわたり存在しているグローバル変数

プログラム中に広範囲にわたって存在しているグローバル変数などは，スライスを抽出すればどの地点で，どのような変更がされているか知ることが非常に簡単になる

1. の場合，まずスライスを抽出する際に変数の指定を行い，その後スライス結果を利用してフォールトを探しはじめる．しかし，フォールト原因となっている変数がすぐに特定できない場合，スライスを用いてフォールト位置特定を行った場合と，用いなかった場合であまり特定時間に差が見られなくなってしまう．2. 3. の特徴は，プログラムのソースコードに依存するものである．しかし，どちらの特徴も実際のプログラムにおいては頻繁に見られる特徴である．

また，本実験の被験者は

1 . フォールトの認知
テストデータをもとに，フォールトの存在を認識する

2 . フォールトの検索
トレースやスライシングによりフォールトの原因を探す

3 . フォールト位置特定

の手順でデバッグを行っており，スライスを使用してデバッグを行った被験者は，スライシングにより2の作業を非常に効率良く行っている．しかし，スライシングを行うには，スライシング基準を指定する必要があり，上述したフォールトの位置特定が短時間で行えた理由としては，このスライシング基準が短時間で正確に特定できたことが挙げられる．スライシングを行った後は，プログラムの参照範囲が小さくなっているので，スライシングを行わない場合よりも，その後のフォールト位置発見までの作業効率に大きく差が出る．

## 7　あとがき

本研究では，デバッグ作業におけるフォールト位置特定において，プログラムスライシング技法が有効であるかどうかを実験的に評価した．その評価方法として従来のフォールト位置特定作業であるトレース機能を用いた技法と，スライシング機能とトレース機能を併用した技法の比較実験をプログラムスライスを用いたデバッグ支援システム上で行った．その結果，プログラムスライシングがフォールト位置特定に有効であることが確認できた．特にフォールトの症状として現れる誤った値の代入された変数が特定できれば，そこをスライシング基準としてスライスを抽出することにより簡単にプログラムの参照範囲を小さくすることができ，短時間でフォールト位置特定ができる．また，あるグローバル変数がプログラム中の至るところで存在する場合，プログラムスライスによりそのグローバル変数がどの地点でどのように変更されているか，またどの条件文で影響を受けているかを容易に知ることができる．また，プログラム中に類似した手続きやルーチンが存在する場合，本システムでスライスを表示する際その部分との比較を容易に行うことができる．

今後の課題としては，システムのインタフェースの改善がまず挙げられる．特に抽出したスライス部分のみを実行させる機能を充実させれば，システムにおけるフォールト位置の特定時間は更に短縮されることが期待される．

また，スライシングを用いてフォールト位置の特定を行う場合，スライスを短時間で正確に抽出すれば，フォールト位置特定を非常に効率良く行うことができる．そこで，デバッグ作業者にスライス基準の取り方についての知識を向上させる必要がある．このためスライシングの効率的な取り方を示すガイドラインを作成する必要がある．

謝辞

本論文作成にあたり, 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.

本論文作成にあたり, 適切な御指導および御助言を頂きました同 楠本 真二 講師に深く感謝致します.

本論文作成にあたり, プログラムスライスに関する事項および実験の面において多大な御指導および御助言を頂きました同 高田 智規 氏に深く感謝致します.

本論文作成にあたり, 適切な御指導および御助言を頂きました同 神谷 年洋 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.

本論文作成にあたり, 実験の被験者になって頂いた, 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] Agrawal, H. and Horgan, J. R.:“Dynamic Programu Slicing”, Proc. ACM SIGPLAN’90 Conference on Programming Language Desine and Implementation, ACM Press, New York,N.Y.,pp.246–256(1990).

[2] 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).

[3] Korel, B. and Laski, J.:“Dynamic Slicing of Computer Programs”, J. Systems Software, Vol.13,pp.187–195(1990).

[4] Myers, G. J.: The Art of Software Testing, Wiley-Interscience(1979).

[5] Weiser, M.:“Program Slicing”, IEEE Trans. on Soft Eng., Vol.10, No.4, pp.352–357(1984).

[6] Weiser, M.:“Programmers use slices when debugging”, Communications of the ACM, Vol.25,No.7, pp.446–452(1982).

[7] 石村貞夫:“統計解析の話”, 東京図書 P.211–215(1989).

[8] 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依存関係解析とそれに基づくプログラムスライシング”，信学技報，SS93–24, pp. 33–40(1993–09).

[9] 植田良一:“再帰を含むプログラムのスライスを求めるアルゴリズムの提案”, 大阪大学大学院基礎工学研究科修士学位論文 (1994–02).

[10] 芝, 渡部, 石塚 編: 統計用語辞典, 新曜社 (1984).

[11] 佐藤慎一:“プログラムスライスを用いたデバッグ支援システムの試作”, 大阪大学基礎工学部情報工学科 特別研究報告 (1994–02).

[12] 下村隆夫:“プログラムスライシング技術と応用” , 共立出版株式会社, p.1–23 (1995).

[13] 山崎利治:“共通問題によるプログラム設計技法解説”,
情報処理学会誌 , 25, 9, p.934(1984).

付録

1 スライシングシステムマニュアル

2 酒屋問題仕様書

3 プログラムリスト (実験 1，実験 2)

# 付録1. スライシングシステムマニュアル

## 1.1 システムの概要

本システム( tksldb )は，スライシング，実行機能をもったデバッグ支援システムプロトタイプである．

## 1.2 使用法

以下に，システムの基本的な使い方を記す．

### 1.2.1 tksldb の実行

tksldb [filename]

とタイプするとウインドウが一つ起動し，その中で作業を行なうことができる．このウィンドウは以下のような 3 つの小ウィンドウで構成されている．

1 source code ウィンドウ
実際に使用する，プログラムのソースコードが表示される．このウィンドウにはエディタ機能がついており，ソースコードを書き直すことができる．

2 status ウィンドウ
source code ウィンドウのプログラムのステータスが表示される．

3 display ウィンドウ
source code ウィンドウのプログラムの実行結果が表示される．また，入力を必要とするプログラムの入力も，このウィンドウ内で行なう．

ウィンドウ上部には，以下のようなメニューが表示されている．

1 File
使用するプログラムのソースコードや，実行させる時に使用するデータファイル名の指定を行なう．

2 Analyze
使用するプログラムの解析を行なう．

3 Excute
プログラムを実際に実行させる．

4 Option

スライシングにおける設定や，データファイルの使用を設定する．

5 Misc

スライシングシステムを終了させたり，上記のウィンドウをクリアしたりする．

引数にプログラム名を指定すると，あらかじめプログラムを読み込むこともできる．引数を指定しなかった場合には，”File”メニューの中の”load”をクリックしてプログラムを読み込む．”load”を選ぶと新たにウィンドウが 1 つ起動しその中にはカレントディレクトリのディレクトリと < .pas > 拡張子のついたファイルが表示されるので，使用したい部分をダブルクリックしてファイルを選ぶ．なお，ディレクトリをダブルクリックした際は，そのディレクトリについて同様の表示を行なう．

プログラムを読み込めばステータスウィンドウに
「load file [filename]」と表示される．

### 1.2．2 プログラムの解析

ソースプログラムを読み込んだ後に，”Analyze”メニューの中の”Efficient Analyze”を選んで解析を行なう．解析が始まると，status ウィンドウに
「start analyze program」と表示され，解析が終了すると「analyze finished」と表示される．なお，解析は以下の 2 種類を行なう．

1 構文解析 ( コンパイル )
実際にプログラムをコンパイルする．ここでプログラムのコンパイルに失敗すると，status ウィンドウにエラーメッセージを表示する．

2 依存解析
プログラムをスライスするために，PDG ( Program Dependence Graph) を作成するための解析を行なう．

解析が完了すると，マウスカーソルを source code ウィンドウに表示されたソースプログラム上に持っていくと，プログラム中の文の色が変わる．ここでマウスの右ボタンをクリックすると，その文に到達する定義 ( 依存解析により得られた ) が status ウィンドウに表示される．ここで記した定義とは，PDG におけるその文の節点の番号，データ依存辺 ( Data Dependence )，制御依存辺 ( Control Dependence ) である．

### 1.2．3　プログラムの実行

解析が終了すると，実際にプログラムを実行できる．”Execute” メニューの中には，”Run” と”Trace”の二種類をがあり，どちらかを選んで実行できる．

1 Run

”Run”はプログラムを普通に実行させるものである．その実行結果はdisplayウィンドウに表示される．

2 Trace

”Trace”はプログラムをステップ実行させるものである．その実行結果はdisplayウィンドウに表示される．”Trace”を選択すると，新しいウインドウが起動し，ステップ実行が始まる．その中では以下のようなことが行なえる．

(a) Next

現在の文を実行し，次の文に移動する．

(b) Eval Var

変数の値を参照する．変数名はこのウインドウの下部のentryウィジェットに入力する．スペースで区切ることによって複数の変数を同時に入力することができる．また，配列変数については，その変数名のみを入力した場合には，先頭の要素が，添字をvarname[x](xは数字)で指定した場合には，その要素の値を参照できる．

(c) Set Val

変数に値を設定できる．まず，どの変数に入力するのか名前を入力し，Returnを押すと，次にどの値に設定するのか聞いてくるので，値を入力する．

(d) Toggle B.P

ブレークポイントのon/offを行なう．どの節点について行なうのか聞いてくるので，その節点番号を入力する．もう一度入力するとブレークポイントがoffになる．

(e) Cont

ステップ実行ではなく，実行を続けて行なう．この時，実行途中にブレークポイントが設定されていれば，そのブレークポイントのところで実行が止まる．

(f) Quit

実行を中断する．

### 1.2．4　追加機能

- ブレークポイントの設定
  プログラムを Trace 中にマウスカーソルをプログラム上に置いて右ボタンをクリックすると，その文にブレークポイントを設定することができる（もう一度押すと off になる）

- データファイルの使用
  実行時にデータファイルを使用する際，まず”Options”メニューの中の「Read from File」を ON にしておく．つぎに”File”メニューの中の”Data File”をクリックすると，新たにウィンドウが起動する．ここにはカレントディレクトリのディレクトリと，拡張子のついてないファイルが表示される．読み込むデータファイルの決定方法は，ソースコードの決定方法と同じである．読み込みが完了すると，status ウィンドウに「data file [filename]」と表示される．

### 1.2．5　スライシング

マウスカーソルをsource code ウィンドウに表示されたソースプログラム上に持っていくと，プログラム中の文の色が変わる．これにより，スライスの起点となる文を選び，そこで右ボタンをクリックする．すると別のウインドウが起動するので，そこでスライスの設定を行なう．その設定には以下のようなものがある．

1 VarName
どの変数に関するものなのかを示す変数名を指定する．

2 Backward，Forward
backward sliceかforward sliceかを選択する．backward sliceとは，スライスの起点となっている文に対して，影響を与えている部の集合である．forward sliceとは，スライスの起点となっている文が，影響を与えている文の集合である．

3 Control flow，Data flow
PDGのアークには Control flow，Data flowの二種類があり，通常は両方に対して，スライスするので，両方ともonに設定する．そのうちのどちら一方のみを辿りたい場合には，どちらか一方のみをonに設定する．

4 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( backward slice なら初期状態まで，forward sliceなら終了状態まで )辿る．

全て設定できれば，”Ok”ボタンを押す．すると，status ウィンドウに
「calculating slice」と表示され，source code ウィンドウにはスライス部分が網掛けになって表示される．

### 1.2．6　スライシングのオプション

本システムではいくつかのスライシングオプションが設定できる．これは”Options”メニューの中の”Options”で設定できる．オプションには以下のようなものがある．

（[On/Off]は初期状態）

1 Only Slice
実行時にスライス部分のみを実行する．[Off]

2 With Comment
コメント行をスライスに含める．[Off]

3 Global Slice
スライスをグローバルに行なう．[On]

4 Local Slice
スライス時に手続きを越えない．[Off]

5 With Token
予約語(program，begin ,... )などもスライスに含める． [Off]

6 Read from File
データファイルの読み込みの設定． [Off]

## 付録2. 酒屋問題仕様書

### 2.1 要求仕様

ある酒屋では，すべての在庫(酒)を幾つかのコンテナに載せて，一つの倉庫に保管している．受付係」の仕事は，倉庫で在庫の積み下ろしを行う「倉庫係」，及び，客の注文を聞いて客に酒を届ける「配送係」との間で数種類の伝票をやりとりしながら，倉庫内の在庫を管理することにある．

#### 2.1.1 受付係への入力

(I1) 積荷票

酒を積んだコンテナが倉庫に搬入されると，倉庫係から「積荷票」が送られてくる．1台のコンテナに酒を10銘柄まで混載できるため，積荷票には，搬入されたコンテナの番号(コンテナに付けられたシリアル番号)，搬入年月日と共に，積載銘柄数が記入されている．さらに，積載されている銘柄番号(銘柄ごとに付けられたシリアル番号)，銘柄毎の数量(積載量)が積載銘柄数だけ繰り返し記入されている．
尚，同じコンテナ内で同じ銘柄(番号)の酒が別々に積載されているならば，銘柄数は別々にカウントされているものとする．

| 積荷票　コンテナ番号　搬入年月日　銘柄数 |
| --- |
| (銘柄番号，数量)の繰り返し |

(I2) 出庫依頼票

客から酒の注文があると，配送係から「出庫依頼票」が送られてくる．出庫依頼票には，出庫すべき酒の銘柄番号(1銘柄のみ)，数量，及び，送り先の番号(送り先の個人に付けられた顧客番号)が記入されている．

| 出庫依頼票　送り先の番号　銘柄番号　数量 |
| --- |

### 2.1．2　受付係からの出力

(O1) 在庫不足票

出庫依頼票により出庫を依頼されたが，出庫すべき酒が必要量だけ倉庫にないなどの理由により出庫できない場合，配送係に「在庫不足票」が送られる．在庫不足票には，出庫できなかった酒の銘柄番号 (1 銘柄のみ)，数量，送り先の番号，及び，対応する出庫依頼票の依頼番号 (??の (A1) 参照) が記入されている．

| 在庫不足票　依頼番号　送り先の番号　銘柄番号　数量 |
|---|

(O2) 出庫指示票

出庫すべき酒が必要量以上に倉庫にある場合，倉庫係に「出庫指示票」が送られる．複数のコンテナから出庫するような指示も許されるので，出庫指示票には，出庫すべき酒の銘柄番号 (1 銘柄のみ)，送り先の番号，対応する出庫依頼票の依頼番号と共に，出庫の対象となるコンテナの数 (最大 10 個) が記入されている．さらに，コンテナ番号，コンテナ毎の数量 (出庫量)，及び，空コンテナマーク (出庫によってコンテナが空になるかどうかを示すマーク ( 0 ：空，1 ：空でない ) ) が出庫の対象となるコンテナの数だけ繰り返し記入されている．

| 出庫指示票　依頼番号　送り先の番号　銘柄番号　コンテナ数<br>(コンテナ番号，数量，空コンテナマーク) の繰り返し |
|---|

### 2.1．3　受付係の作業

(A0) 在庫の把握

積荷票や出庫指示票に基づいて，倉庫内のコンテナや酒の在庫状況を常に把握する．

(A1) 出庫依頼票への依頼番号の付加

倉庫係から送られてきた出庫依頼票に，受付順に 1 からの通し番号を依頼番号として付加する．依頼番号の付加は，出庫依頼票が送られてきた時点で行う．

(A2) 在庫不足票の作成

出庫依頼票に依頼番号を付加した時点で，出庫依頼票で出庫するよう指定された酒が必要量だけ倉庫にない場合，在庫不足票を作成し，配送係に送ると共に，その控え (「在庫不足票控」) を作成し，保管する．ただし，指定された酒と同一銘柄の酒に対する在庫不足票控が存在する場合には，在庫の有無に関わらず在庫不足票を作成し，配送係に送ると共に，その控えを作成し，保管する．なお，在庫不足票控は在庫不足票と同一形式で，その記入内容は対応する在庫不足票と同じとする．

(A3) 出庫指示票の作成

出庫依頼票に依頼番号を付加した時点で，出庫依頼票で出庫するよう指定された酒と同一銘柄の酒に対する在庫不足票控が存在せず，かつ，指定された酒が必要量以上に倉庫にある場合，出庫指示票を作成し，倉庫係に送る．また，在庫不足票控が存在する場合には，新たに在庫が搬入された (積荷票が倉庫係から送られてきた) 時点で在庫を確認し，在庫不足票控で指定された酒が必要量以上に倉庫にある場合，出庫指示票を作成し，倉庫係に送る．

なお，出庫指示票の作成においては，次のような制約条件がある．

- 出庫の対象となるコンテナが複数の場合には，コンテナ番号の小さいものから順に記入されているものとする．
- 出庫の対象となるコンテナ数が 10 を越える場合には，コンテナ番号の小さいものから順にコンテナを 10 個ずつのグループに分け，グループ毎に出庫指示票を作成するものとする．
- 在庫不足票控が複数存在し，かつ，新たな在庫搬入によって同時に複数の出庫指示票が作成される場合には，対応する在庫不足票控の依頼番号の小さいものから順に出庫指示票を作成するものとする．
- 在庫不足票控で指定された酒が必要量以上に倉庫にある場合であっても，その依頼番号よりも小さな依頼番号を持つ同一銘柄に対する在庫不足票控が存在する場合には，出庫指示票を作成することはできないものとする．

(A4) 在庫不足票控の破棄

在庫不足票控に基づいて出庫依頼票を作成し，倉庫係に送った場合には，対応する在庫不足票控をその時点で破棄する．

### 2.2 外部仕様

プログラムへの入力は「積荷票」と「出庫依頼票」の時系列であり、出力はそれに基づいて作成された「在庫不足票」と「出庫指示票」の時系列である。

### 2.3 sakayaの入出力データフォーマット

[入力データフォーマット]

積荷票... 票識別子、コンテナ番号、搬入年月日、銘柄数からなる1つのレコードと、銘柄番号、数量からなる1つ以上のレコードで構成される。

| データ項目 | 票識別子 “t” | コンテナ番号 | 搬入年月日 | 銘柄数 |
|---|---|---|---|---|
| 属性 | 文字定数 | 整数 | 整数 | 整数 |
| 桁数 | 1 | 5 | 6 | 2 |

| データ項目 | 銘柄番号 | 数量 |
|---|---|---|
| 属性 | 整数 | 整数 |
| 桁数 | 4 | 4 |

注：コンテナ番号は「1」からはじまる5桁の整数である。
搬入年月日は、上位2桁が西暦年の下位2桁を、中位2桁が月(1 ~ 12)を、下位2桁が日(1 ~ 31)を、それぞれ表す。
上記の銘柄数、銘柄番号、数量の桁数は、最大の桁数である。

積荷票の例
コンテナ番号が「10000」、搬入年月日「1996年8月23日」、銘柄番号「1」の酒が「20本」、銘柄番号「2」の酒が「20本」つまれているならば、積荷票には以下のように書かれている。

```
t 10000 960823 2
1 20
2 20
```

出庫依頼票... 票識別子、送り先の番号、銘柄番号、数量からなるレコードである。

| データ項目 | 票識別子 “s” | 送り先の番号 | 銘柄番号 | 数量 |
|---|---|---|---|---|
| 属性 | 文字定数 | 整数 | 整数 | 整数 |
| 桁数 | 1 | 4 | 4 | 4 |

注：送り先の番号は「1」からはじまる４桁の整数である。
上記の銘柄番号、数量の桁数は、最大の桁数である。

出庫依頼票の例
顧客番号「1234」の客に、銘柄番号「1」の酒を、「10本」依頼されたら、出庫依頼票には以下のように書かれている。

```
s 1234 1 10
```

[出力データフォーマット]

在庫不足票... 出力メッセージからなる2種類のレコードと、依頼番号、送り先の番号、銘柄番号、数量からなる1つのレコードで構成される。

| データ項目 | 出力メッセージ<br>“%%%%%%% OUTPUT %%%%%%%%%%%%”<br>“zaiko fusoku hyou” |
|---|---|
| 属性 | 文字定数 |
| 桁数 | 44 |

| データ項目 | 出力メッセージ<br>“no.　　name　　sake　　ryou” |
|---|---|
| 属性 | 文字定数 |
| 桁数 | 32 |

| データ項目 | 依頼番号 | 送り先の番号 | 銘柄番号 | 数量 |
|---|---|---|---|---|
| 属性 | 整数 | 整数 | 整数 | 整数 |
| 桁数 | 5 | 4 | 4 | 4 |

在庫不足票の例

```
%%%%%%% OUTPUT %%%%%%%%%%%%
zaiko fusoku hyou
no.      name        sake      ryou
1        1234          1         10
```

上記の在庫不足票は、依頼番号「１」、顧客番号「１２３４」、不足している酒の番号「１」、不足量「１０本」を表している。

出庫指示票... 出力メッセージからなる3種類のレコード、依頼番号、送り先の番号、銘柄番号、コンテナ数からなる1つのレコード、コンテナ番号、数量、空コンテナマークからなる1つ以上のレコードで構成される。

| データ項目 | 出力メッセージ<br>“<<<<<OUTPUT<<<<<<<<<”<br>“syukko sizi hyou” |
|---|---|
| 属性 | 文字定数 |
| 桁数 | 35 |

| データ項目 | 出力メッセージ<br>“no.　　name　　sake　　num.kontena” |
|---|---|
| 属性 | 文字定数 |
| 桁数 | 41 |

| データ項目 | 依頼番号 | 送り先の番号 | 銘柄番号 | コンテナ数 |
|---|---|---|---|---|
| 属性 | 整数 | 整数 | 整数 | 整数 |
| 桁数 | 5 | 4 | 4 | 2 |

| データ項目 | 出力メッセージ<br>“kontena　suu　　mark” |
|---|---|
| 属性 | 文字定数 |
| 桁数 | 20 |

| データ項目 | コンテナ番号 | 数量 | 空コンテナマーク |
|---|---|---|---|
| 属性 | 整数 | 整数 | 整数<br>空なら “0”、空でなければ “1” |
| 桁数 | 5 | 4 | 1 |

出庫指示票の例

```
<<<<<OUTPUT<<<<<<<<<
syukko sizi hyou
no.        name        sake      num.kontena
1          1234         1          1
kontena  ryou    mark
10000      10       0
```

上記の出庫指示票は、依頼番号「１」、依頼した客の番号「１２３４」、出庫する酒の番号「１」、出庫に必要なコンテナ数(出庫指示票に記載されたコンテナの種類の数)「１」、そして出庫にあたっての、当該コンテナのコンテナ番号「１００００」、出庫

量「１０本」、そのコンテナが、この出庫指示により空になることを示す、空コンテナマーク「０」が記されている。

注）各データ項目は1つ以上の空白で、各レコードは1つの改行記号で、各伝票は1つの改行記号で、それぞれ区切られるものとする。さらに、票識別子に'q'を入力するとプログラムが終了し、's、t、q'以外の文字を入力すると'ERROR'というエラーメッセージを出力する。