ソフトウェア工学　136－22
（２００２．３．８）

探索型デバッグ手法におけるログの特徴値の利用

田村 文隆　　岡本 渉　　植木 克彦　　平山 雅之
株式会社東芝 研究開発センター
Email: tamu@sel.rdc.toshiba.co.jp

探索型デバッグ手法は、ユーザが効率良くバグの原因箇所を特定する手助けを行う手法である。これを実現する為に、我々は、特徴値、類似度と言う二つの量を定義し利用している。本稿では、特徴値を中心に、その意味、利用方法、定め方を述べる。

特徴値は、ログ内に含まれる各事象に対し、その発生確率が低い程大きい値を取る様に定められる。但し、この定義は、多数の事象が同じ特徴値を持つと言う縮退の問題を持つ。我々は、この縮退を解く方法を考察した。

最後に、現在我々が進めている実験の一例を紹介する事により、特徴値の定め方の有効性を示す。

Applying Peculiarities of Traces in Exploratory Debugging

Humitaka Tamura　Wataru Okamoto　Katsuhiko Ueki　Masayuki Hirayama
TOSHIBA corp. R&D Center

We proposed ‘Exploratory Debugging Method’ which supports users to locate the failures in software. For the quantitative analysis, we defined two values: ‘peculiarity’ and ‘similarity’. In this paper, we focus the former and explain its details.

Peculiarities are defined for each event in a given program’s trace, such that rare events have high values. However, there are high degeneracies in the peculiarities defined above. We studied a method to resolve this degeneracy.

To illustrate the adequacy of our method, we show one practical example chosen from our experiments.

1　はじめに

近年のソフトウェア規模の増大、複雑化に伴い、テスト、デバッグに要する費用は年々増加してきている。我々は、テスト作業の効率化、及び、デバッグ作業の効率化それぞれに焦点を絞り、トータルとしての費用を削減する事を目標にして研究を進めている。本稿では、特に、後者を目的として、我々が提案している探索型デバッグ手法 [1, 2, 3, 4] における特徴値の扱いについて、特にその中で利用している「特徴値」と言う概念を中心に報告する。

ソフトウェアに含まれる多くの不具合の原因 (欠陥) は、正常動作時とは異なる振舞いに含まれている事が多い。探索型デバッグ手法の特徴は、この性質を利用する点に有る。特に、ソフトがある程度の完成度に達し、稀なケースでしか不具合が生じなくなってからのデバッグは非常に時間を要する。探索型デバッグ手法はこの段階において威力を発揮する。

この手法では、テスト時にプログラムの動作記録 (ログ) を保存する。そして、異常動作時のログと正常動作時の各ログとを比較し、異常動作時のログ中の特徴的な振舞いを見い出す。これにより、欠陥の含まれている可能性の高い箇所に優先付けを行ってユーザに示す。又、異常動作時のログに似た特徴を持つ正常動作時のログを見つけ、両者の違いを示して、不具合の原因究明のサポートをする。

上記を定量的に行う為、我々は特徴値と類似度と言う二つの量を定義する。特徴値は一つのログ、及び、そのログ中に含まれる各種のイベント (どの関数を実行したか、どの分岐を通ったか等) に対して定められる。これに対し、類似度は二つのログ間に対して、及び、二つのログ中に含まれるイベント間に対して定められる。

我々が探索型デバッグ手法を発案し、ツールとしてインプリメントした当初は、類似度の特別な場合、即ち、ログとそのログ自身、又は、イベントとそのイベント自身との間の類似度を、そのログ又はイベ

ントの特徴値として定義していた [3]。今回、この定義の見直しを行い、より正確にログ中の特徴的な振舞いを見い出す事が出来る様になったので、本稿ではこれを報告する。

以下、第 2 章では、特徴値の定義に先立ち、まず、これらをどの様に利用するかを紹介した後、第 3 章で、具体的な計算方法を示し、第 4 章で実際の適用例を紹介して行く。尚、類似度については、第 2 章で利用法のみを紹介した。類似度の定義、計算方法については紙面の都合上省略した為、[3] を参照されたい。

2 特徴値の利用

ここでは、探索型デバッグ手法の手順を追いながら特徴値の概念を紹介する。

本手法の大まかな流れは以下の通りである。

1. **実行ログの記録** プログラムの実行時の動作 (正常動作時、異常動作時) で生じたイベントをログに記録する。
2. **ログの切り出し** 比較起点となる位置をユーザに指定させ、記録したログから起点以下の部分ログを切り出す。
3. **ログの分析** 切り出された各ログに含まれるイベントを比較し、それらのイベントの特徴値、イベント間の類似度、及び、各ログの特徴値、ログ間の類似度を定める。
4. **ソースコードの調査** 得られた特徴値、類似度を元に、疑わしいログ、ログ中の疑わしい位置を選び、それに対応するソースコードを優先的に調査する様ユーザに示す。

例えば、リスト 1 に示したプログラムで、入力 j に 0,1,2 を与え、三つのログを得た場合を考える。不具合は j=0 の時に生ずるとする。

```
void main() {              void f(int i,j) {
  int i,j;                    ...
  input j;                    p();
  for (i=0;i<3;i++) {         q();
    f(i,j);                   ...
    g(i);                  }
  }
  f(5,j);                  vod g(int i) {
  f(6,j+1);                   ...
}                          }
```

リスト 1: プログラム例

1. のログの記録ステップで記録されるイベントとは、プログラムが実行した位置や関数、ループ、分岐等を意味する (実行時のデータまで考慮した場合の拡張については [4])。ここで記録されたログは、根付き木構造を持つ様に整形される。木の各ノードは記録された各イベントに対応し、ある関数内で実行されたイベントは、その関数の子ノードになると言った性質を持っている。

2. のログの切り出しステップでは、ターゲットプログラムに対するユーザの知識、デバッグの進行状況等により比較起点の選択の仕方が異なって来る。もし、不具合の原因 (これは、不具合が観測された位置に有るとは限らない) に全く見当が付かない場合には、比較起点を関数 main() とする。これによって、入力 j=0,1,2 に対する三つのログがそのまま切り出されて、それぞれが比較の対象となる。

一方、プログラムの動作にある程度知識が有り、例えば、不具合の原因が、関数 f() 内、又は、f() 内部から呼び出されている関数内に有ると信ずる根拠が有るなら、比較起点を関数 f() とすれば良い。この場合には、三つのログそれぞれから、関数 f() 以下の部分ログ五つが切り出され、計 15 の部分ログが比較対象となる。同様に、for ループを実行している時に問題が生じていると考えるならば、ループ個々の実行を比較起点にして、各ログから四つの部分ログ (i==3 を含む) を切り出し、計 12 の部分ログを比較対象とする事も出来る。

3. のログの分析ステップでは、2. で切り出された各ログに含まれるイベント間に対応付けを行い、それらを比較する事によって各イベントの特徴値、イベント間の類似度を定める。これらは、そのイベントの出現確率、即ち、切り出されたログの中でそのイベントに等価なイベントがどれだけ有るか、を元にして決定される。詳細は第 3 章で述べる。

4. のソースコード調査ステップには幾つかの形態が考えられる。ここでは、典型的な二例を紹介する。

[**イベントの特徴値を利用**] 比較起点を main() にし、三つのログを比較対象として切り出した場合を考える。この場合、不具合の生じたログ (入力 j=0) の中の疑わしい部分を、図 1(a),(b) に示した様な特徴値の調査結果を元にして優先的に調査すれば良い。同図 (a) は、main() 直下のイベントの中では f(5,0) が最も高い特徴値を持ち、この内部を調査すると不具合の原因を見つけ出せる可能性が高い事を意味する。f(5,0) を展開した時には、同図 (b) の様に、再び二つの関数がイベントとして現れ、その中では、関数 p() 内部の方が疑わしい事が分かる。

我々のツールは、この様にユーザに疑わしいイベントを順に展開して調べて行かせる事により、異常動作したログ中の特別な振舞いを知らせると共に、その特別な振舞いに至る経路を理解させる。しかし、上

の情報だけでは、ユーザは、その振舞いがなぜ異常ログで特別なのか理解出来ない。これをユーザに教える為に、ログ間の類似度を利用して、切り出されたログの中から、異常動作したログに似たログ (両者間の類似度が大きな類似度を持つログ) を選び、両者の違いを表示する (図 2)。この図を見れば、異常動作したログに最も似た正常なログは、入力 j=2 のログで、両者の違いは、関数 p() 内から関数 r() を呼ぶかどうかに有る事が分かる。

[**ログの特徴値、イベントの特徴値を共に利用**] ユーザにプログラムの動作に対する知識が有り、ステップ 2. で比較の起点として関数 f() を指定した場合を考える。この場合には、比較の対象となるログ 15 個の内、異常動作するログに関する物が 5 個有る。これら 5 個のログを、ログの特徴値の大い順にユーザに示せば、ユーザは異常動作の原因を含む可能性の高いログから順に調査して行く事が出来る。選んだログの中で、特徴値の高いイベントから調査して行く方法は、先に述べた方法と同様にして行える。

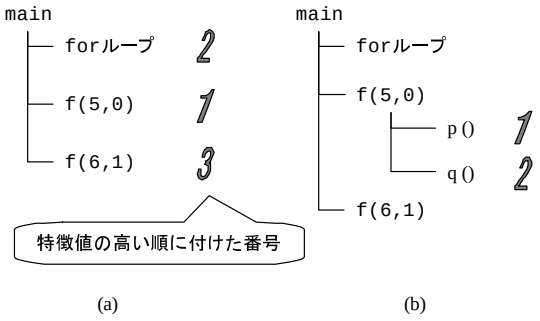

図 1: 特徴

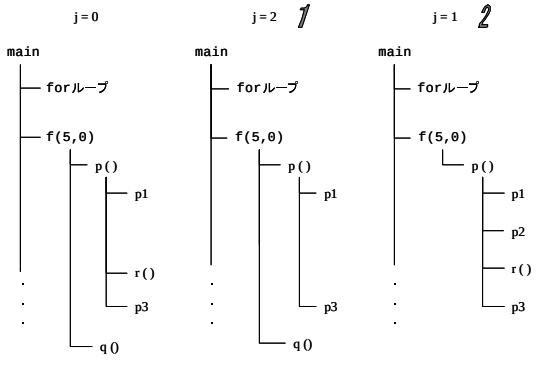

図 2: 比較

3 特徴値の計算

ここでは、切り出されたログの集合から、それぞれのログの特徴値、ログ内のイベントの特徴値を定める方法を示す。

3.1 単純な場合

まず、切り出されたログ内に、新たな関数呼び出しや分岐、ループ等の内部構造が全く無い場合を考える。これは、例えば、リスト 2 の cond1 や cond2 の分岐文を比較の起点として、ログを切り出した場合に相当する。

```
if ( cond1 ) {          if ( cond2 ) {
  /* blk1 */              /* blk3 */
  i = 1;                  i = 1;
} else {                  j = 2;
  /* blk2 */              k = 3;
  i = 2;                  l = 4;
  j = 3;                }
}
```

リスト 2: プログラム例

今、cond1 の分岐文, cond2 の分岐文いずれを起点にしても 10 個づつログが切り出され、かつ、どちらも 10 個のログの内一つだけが条件を満たすとする。この時、我々は、cond1 を満たすログも、cond2 を満たすログも同じ特徴値を取る様に特徴値を定めたい。この為には、各行ごとの実行を別々のイベントと捉えるのではなく、論理的なブロックの実行全体を一つのイベントとして扱えば良い。又、cond2 の分岐で条件が満たされなかった時の様に、「分岐で何も行われ無かった」事も一つのイベントとして扱う。従って、以下では、cond1 の分岐で切り出したログの内、1 個のログでイベント {blk1}、9 個のログでイベント {blk2} が発生しており、一方、cond2 の分岐で切り出したログでは、1 個のログでイベント {blk3}、9 個のログでイベント {NULL} が発生していると考える。

一般にイベント e の特徴値 $T(e)$ は、そのイベントの発生確率 $P(e)$ を元にして、その情報量で定義する: $T(e) = -\ln P(e) \geq 0$. 今の場合、論理ブロックに関するイベント {blk1}, {blk2} の発生確率はそれぞれ 1/10, 9/10 なので、特徴値は、$T(\{\texttt{blk1}\}) = -\ln(1/10) \simeq 2.30$, 又、$T(\{\texttt{blk2}\}) = -\ln(9/10) \simeq 0.11$ となる。これは、blk1 を実行するイベントの方がより珍しい事を意味している。cond2 についても全く同様となる。

上の考え方自体は特に新しい物ではなく、Web 上の全文検索エンジンで、複数のキーワードが指定された時、ありふれた単語の重みを低くして、より希少なキーワードの一致を優先すると言った手法 (例えば [5]) 等でも利用されている。

ログ L の特徴値 $U(L)$ は、今の単純な場合には、ログ内の唯一のイベントの特徴値その物で定める。従っ

て、cond1 を満たすログの特徴値は $U(\text{true}) \simeq 2.30$, その他九つのログの特徴値は皆 $U(\text{false}) \simeq 0.11$ となる。

3.2 一般の場合

次に、切り出されたログが複雑になり、ログ内に関数呼び出し等の内部構造 (それ自身、更に内部構造を持ち得る) が含まれる一般の場合に定義を広げる。

一般の場合のログの特徴値は、そのログ内に含まれる全イベントの特徴値の最大値として定義する。つまり、ログの中に現れる最も珍しいイベントの持つ特徴値が、そのログの特徴値となる。ここで言うイベントは、前節と同様、論理ブロックに関するイベントだけを指す。但し、同じ論理ブロックに関するイベントでも、比較の起点から異なる経路を辿って実行された物は別イベントとして扱う。

```
void f(int i) {
  /* f1 */
  g(i);
  if ( i < 2 ) {
    /* f2 */
    g(i+1);
    h();
  } else {
    /* f3 */
    h();
  }
}
```

```
void g(int i) {
  /* g1 */
  if ( i < 1 ) {
    /* g2 */
    printf("g2");
}
void h() {
  /* h1 */
  return;
}
```

リスト 3: プログラム例

具体的に、リスト 3 に示した関数 f() を起点にして切り出した三つのログ L0, L1, L2 を考える (図 3)。各ログは f() の引数 i に 0,1,2 を与えて得られる。

この図で、例えば、論理ブロック g1 に関するイベント {g1} としては、

- 各ログの先頭から {f1}→g()→{g1} の順に実行される物、
- ログ L0,L1 の先頭から、{f1}→ 分岐 →{f2}→ g()→ {g1} の順に実行される物

の二種類が有る。我々はこれらを異なるイベントとして考える。前者のイベントは、三つのログ全てで発生しているので、その特徴値は $-\ln(3/3) = 0$、後者のイベントは L0,L1 だけでしか発生していないので、特徴値は $-\ln(2/3) \simeq 0.4$ となる。

同様にして考えると、L0 中のイベント:{f1} →g() → ··· → {g2} は、L0 でしか発生していないので、その特徴値は $-\ln(1/3) \simeq 1.1$ である。又、L2 中には、{f1}→ 分岐 →{f3}、及び、{f1}→ 分岐 → ··· →{h1} の二つのイベントが L2 だけでしか発生していない。従って、これらのイベントの特徴値も $-\ln(1/3) \simeq 1.1$ となる。これに対し、L1 に含まれるイベントは、L0,L1,L2 全てに含まれる (特徴値 0) か、又は、これらの中の二つのログに含まれる (特徴値 $-\ln(2/3) \simeq 0.4$) 物しか存在しない。ログの特徴値は、内部のイベントの特徴値の最大値とするので、結局、$U(\text{L0}) = U(\text{L2}) = -\ln(1/3) \simeq 1.1, U(\text{L1}) = -\ln(2/3) \simeq 0.4$ と定められる。

次に、関数等の構造を持ったイベントの特徴値の定義を見る。これらのイベントの特徴値も、ログの特徴値と同様、その構造を根とする部分木内に含まれる全ブロックに関するイベントの特徴値の最大値に

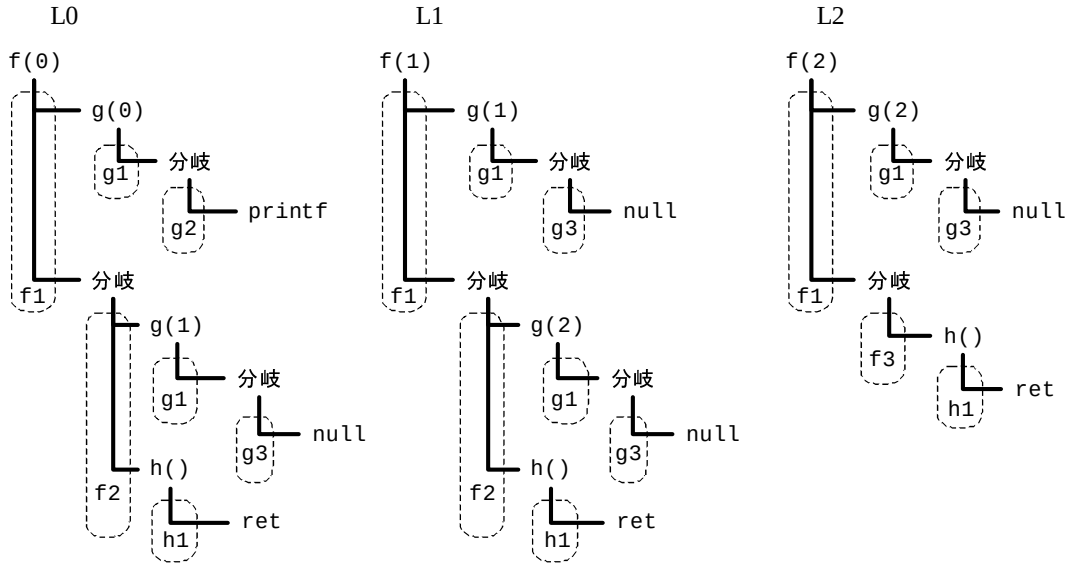

図 3: ログ

より定義する。例えば、L0 のブロック f1 における関数呼び出し g() の特徴値 $T(\mathtt{g}())$ は次で定められる: g(0) を根とする部分ログ内には特徴値 0 のイベント {f1} →g() →{g1} と特徴値 $-\ln(1/3)$ のイベント {f1} → g()→ ⋯ → {g2} が有るので、$T(\mathtt{g}()) = -\ln(1/3)$。一方、L0 のブロック f1 における「分岐」内部には $0, -\ln(2/3)$ の特徴値を持つイベントしか無いので $T(\text{分岐}) = -\ln(1/3)$ である。

以上の例から分かる様に、比較対象となるログの数が少ししか無い場合、同じ特徴値を取るログ、イベントが増えてしまう。比較対象のログが N パターンに分類される時には、特徴値のバリエーションは、N 通り:$-\ln(1/N), -\ln(2/N), \cdots, -\ln(N/N) = 0$ しか存在しない。ユーザには、特徴値最大のログ、イベントを調査する様に指示すべきなので、これらが多く現れるのは問題である。この特徴値の縮退を解く為に、第 3.4 節では、特徴値の定義に深さによる補正を加える方法を紹介する。

3.3 特徴値の計算

以下、前節で定義した特徴値の計算方法を示す。

比較対象として切り出されたログの集合を S, 特徴値を求めるログを L とする。L 直下には、一つだけ論理ブロックが有る事を仮定し、これを $K(L)$ と記述する。又、L 直下に有る構造を $e_1, e_2, \cdots$ とする時、$e_1, e_2, \cdots$ を根とする L の部分木を $L_{e_1}, L_{e_2}, \cdots$ と書く。例えば、前節の図 3 では、$S = \{\mathtt{L0}, \mathtt{L1}, \mathtt{L2}\}$ で、$L = \mathtt{L1}$ とすると、$K(L) = \mathtt{f1}$ である。又、L0 直下には、構造 $e_1 = \mathtt{g}()$、及び、$e_2 =$「分岐」が有る。

前節の定義により、ログ $L \in S$ の特徴値は、L 直下の論理ブロックに関する特徴値と、L 直下の各構造の特徴値の最大値と言う事が出来る:

$$U_S(L) = \max\left[\, T_S(\{K(L)\}),\ \max_{e \in L\text{直下の構造}} T_S(e) \,\right]. \tag{1}$$

上では、U, T の添え字として集合 S を陽に記述した。S の中で $\{K(L)\}$ と同じイベント (前節に述べた様に、同じブロックの実行でも経路が異なれば別のイベントとして扱う) が現れるログの数を n とすると、$T_S(\{K(L)\}) = -\ln \Pr(\{K(L)\}) = -\ln(n/|S|)$ である。又、L 直下の各構造 e に対し、部分ログ L_e 内の全論理ブロック (直下の論理ブロックだけでなく、下位に有る構造の論理ブロックも全て含む) を b1,b2,⋯ とすると、e の特徴値 $T_S(e)$ は、

$$\begin{aligned} & T_S(e) \\ &= \max(-\ln \Pr(\{\mathtt{b1}\}), -\ln \Pr(\{\mathtt{b2}\}, \cdots)) \\ &= -\ln \min(\Pr(\{\mathtt{b1}\}),\ \Pr(\{\mathtt{b2}\}),\ \cdots) \end{aligned} \tag{2}$$

である。尚、確率は、母集団を S として求める点に注意されたい。例えば、図 3 の L0 直下の構造 g() 以下には、二つの論理ブロック g1, g2 が有るので、

$$\begin{aligned} T_{\{\mathtt{L0},\mathtt{L1},\mathtt{L2}\}}(\mathtt{g}()) &= -\ln \min(\Pr(\mathtt{g1}), \Pr(\mathtt{g2})) \\ &= -\ln \min(3/3, 1/3). \end{aligned}$$

以下、$\min(a, b, c, \cdots)$ を図 4(a) の様に表現する。この記法により、式(2) は同図 (b) の様に書かれる。

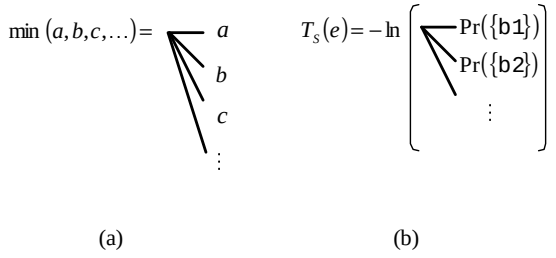

図 4: 図表現 1

ログ L 直下に有る構造 e の特徴値 (式(2)) について、更に考察を進める。

部分ログ L_e 内に含まれる任意の深さに有るブロック b が実行される確率 $\Pr(\{\mathtt{b}\})$ は、

$$\Pr(\{\mathtt{b}\}) = \Pr(e) \cdot \Pr(\{\mathtt{b}\} \,|\, e) \tag{3}$$

と書き換えられる。ここで、$\Pr(e)$ は、S のログが構造 e を含む確率、$\Pr(\{\mathtt{b}\} \,|\, e)$ は、S のログが構造 e を含む条件下での b が実行される条件つき確率である[1]。従って、式(2) は次の様に書き換えられる:

$$\begin{aligned} & T_S(e) \\ &= -\ln[\Pr(e) \cdot \min(\Pr(\{\mathtt{b1}\} \,|\, e),\ \Pr(\{\mathtt{b2}\} \,|\, e), \cdots)] \\ &= -\ln \Pr(e) \\ &\quad -\ln \min(\Pr(\{\mathtt{b1}\} \,|\, e),\ \Pr(\{\mathtt{b2}\} \,|\, e), \cdots). \end{aligned} \tag{4}$$

以下では、$a \cdot \min(b, c, \cdots)$ を図 5(a) の様に表現する。これにより、式(4) は同図 (b) の様に書かれる。

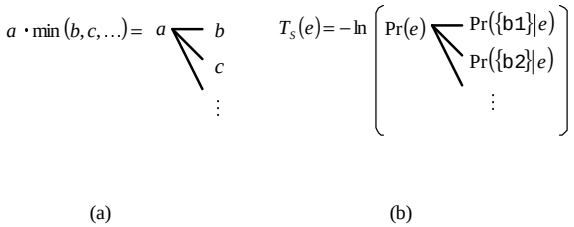

図 5: 図表現 2

[1] 一般に、確率変数 $X, Y, Z, \cdots$ の確率分布を $P(X, Y, Z, \cdots)$ とすると、$P(X) = \sum_{y \in Y, z \in Z, \cdots} P(X, Y, Z, \cdots)$, $P(X, Y) = \sum_{z \in Z, \cdots} P(X, Y, Z, \cdots)$. 又、条件付き確率は、$P(X|Y) = P(X, Y)/P(Y)$, $P(X|Y, Z, \cdots) = P(X, Y, Z, \cdots)/P(Y, Z, \cdots)$ の様に定義される。

今、集合 S から、直下に構造 e を含むログだけを選び出し、それぞれの e を根とする部分ログよりなる集合 S_e を作る (この時 $L_e \in S_e$)。すると、式(3) の $\Pr(\{\mathtt{b}\}|e)$ は、S_e のログが論理ブロック b を実行する確率 (但し、母集団は S_e として計算する) と考える事が出来る。従って、前節に述べた特徴値の元々の定義により、式(4) は、更に次の様に変形する事が出来る:

$$T_S(e) = -\ln \Pr(e) - U_{S_e}(L_e). \tag{5}$$

式(5) の $U_{S_e}(L_e)$ を、式(1) を再適用して計算し、更にその内部を式(5) で展開すると言う再帰計算を、内部構造が無くなるまで続ける事によって、最終的にログ L の特徴値 $U_S(L)$ は、図 6 によって定められる。この図では、L 直下に構造 $e_1, e_2, \cdots$ が有り、構造 e_k $(k = 1, 2, \cdots)$ の下には構造 $e_{k1}, e_{k2}, \cdots$ が有るとした。また、$K(e)$ は構造 e の直下に有る論理ブロック (常に一つ有ると仮定する) を意味する。

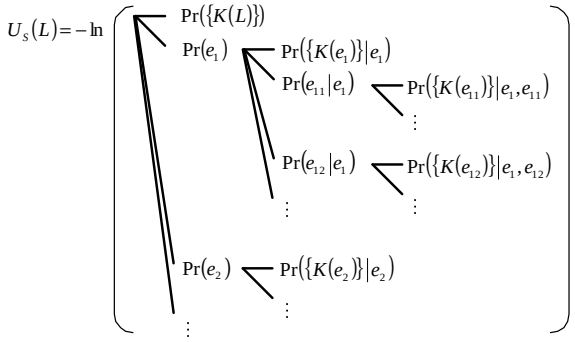

図 6: ログの特徴値

3.4 深さによる補正

以上に示した特徴値の定義は、第 3.2 節の最後に述べた様に、比較対象のログの数が少ないと、多くのログの特徴値が同じ最大値を取ってしまうと言う縮退の問題を含んでいる。そこで、ここでは

1. 根に近い所に有る特徴的な振舞いを優先
2. 根から遠い所に有る特徴的な振舞いを優先

のいずれかによって縮退を解く方法を紹介する。前者は、根に近い所に有る振舞いの方が、ユーザに取って早く理解する事が出来ると言うヒューリスティクスに基づいており、後者は根から遠い所に有る振舞いは、デバッガ等を利用した従来のデバッグ手段のみでは見出すのが難しい為、我々の手法がより有効になると言うヒューリスティクスに基づいている。

縮退を解く為には、深さに関する補正項 $\phi = 1 + \alpha$, $(|\alpha| \ll 1)$ を導入し、図 4(a),5(a) を図 7 の様に修正する。後で見る様に、補正項は、$\alpha < 0$ なら根に近い振舞いを優先する様に働き、$\alpha > 0$ なら根から遠い振舞いを優先する様に働く。

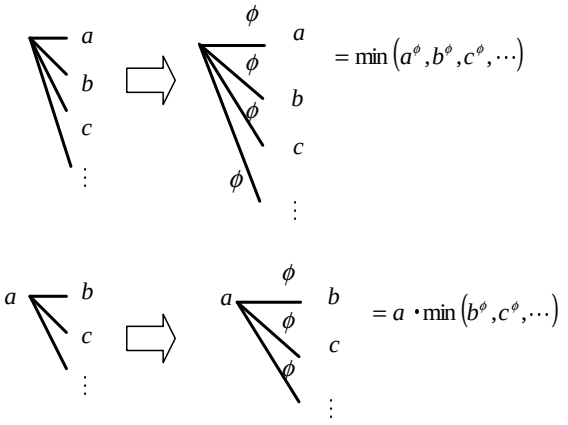

図 7: 補正項の導入

図 6 にこの補正を適用して求めた特徴値 $U_S^{(\phi)}(L)$ について考える。ログ L の根から深さ d(根の深さを 0 とする) に有る論理ブロック bl に関するイベントは、特徴値 $U_S^{(\phi)}(L)$ を求める式に、

$$\begin{aligned} &T^{(\phi)}(\{\mathtt{bl}\}) = \\ &-\ln[\ \Pr{}^{\phi}(e_k) \cdot \Pr{}^{\phi^2}(e_{kl}|e_k) \cdot \Pr{}^{\phi^3}(e_{klm}|e_k, e_l) \cdot \\ &\qquad \cdots \\ &\qquad \Pr{}^{\phi^{d-1}}(e_{\underbrace{klm\cdots n}_{d-1 \text{ times}}}|\underbrace{e_k, e_l, e_m, \cdots}_{d-2 \text{ times}}) \cdot \\ &\qquad \Pr{}^{\phi^d}(\{\mathtt{bl}\}|e_k, e_l, e_m, \cdots, e_n)\ ] \end{aligned}$$

の形で現れる。これを α $(\phi = 1 + \alpha)$ の一次まで残して整理すると、

$$\begin{aligned} &T^{(\phi)}(\mathtt{bl}) \simeq \\ &-\ln \Pr(\{\mathtt{bl}\}) \ + \ d\,\alpha\,(-\ln \Pr(\mathtt{bl})) \\ &\qquad\qquad - \ (d-1)\alpha\,(-\ln \Pr(e_k)) \\ &\qquad\qquad - \ (d-2)\alpha\,(-\ln \Pr(e_{kl}|e_k)) \\ &\qquad\qquad - \ \cdots \\ &\qquad\qquad - \ \alpha\,(-\ln \Pr(e_{klm\cdots n}|e_k, e_l, e_m, \cdots)) \end{aligned} \tag{6}$$

と表される。上式各項の意味を図 8 を例に説明する。

図 8 では、論理ブロック b1,b2,b3 の発生確率は皆 1/4 であり、b4 の発生確率だけ 1/3 である場合を現している:$\Pr(\{\mathtt{b1}\}) = \Pr(\{\mathtt{b2}\}) = \Pr(\{\mathtt{b3}\}) = 1/4$, $\Pr(\{\mathtt{b4}\}) = 1/3$。式(6) の初項 (補正無しの項) によって、b4 に関するイベントの特徴値はその他の特徴値に比べ小さくなる。α は十分小さく取るので、補正項を加えてもこの関係は変わらない。

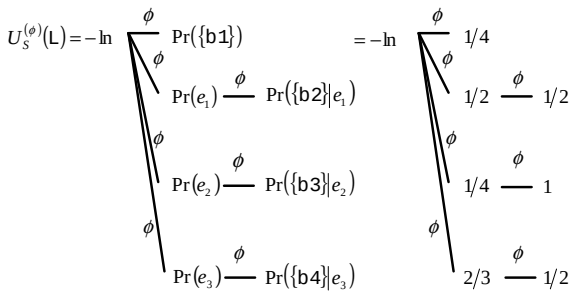

図 8: 例

前節までの補正無しの定義では、b1,b2,b3 に関する特徴値が皆 $-\ln(1/4)$ で縮退する。これに、式(6) 第 2 項と、第 3 項以降の補正が加わって、縮退が解かれる。この内、第 2 項の補正は、イベント {b2},{b3} が、イベント {b1} よりも深い位置で発生している事を表す。従って、この補正項のみなら、{b2},{b3} の特徴値は同じ割合だけ深さに影響される。ここで、イベント {b3} は確かに深さ 2 に有るが、実際に珍しいのは、深さ 1 に有る構造 e_2 の存在である事に着目する必要が有る。構造 e_2 が存在する事が分かっていれば {b3} が発生するのは珍しい事象では無い。つまり、深い位置に有るイベントを優先するならば、本来 {b2} が最も大きい特徴値を持つべきで、逆に浅い位置に有るイベントを優先するならば、{b1},{b3} が大きい値を持つべきである。式(6) の第 3 項以降はこの効果を特徴値に与える。実際 {b1}, {b2}, {b3} の特徴値を式(6) を使って求めると、

$$\begin{aligned} T^{(\phi)}(\{\mathtt{b1}\}) = T^{(\phi)}(\{\mathtt{b3}\}) &\simeq -\ln\frac{1}{4} + \alpha\ln\frac{1}{4} \\ T^{(\phi)}(\{\mathtt{b2}\}) &\simeq -\ln\frac{1}{4} + \frac{3}{2}\alpha\ln\frac{1}{4} \end{aligned}$$

となっている事が分かる。従って、今の場合

$$\text{if } \alpha > 0 \ : \ T^{(\phi)}(\{\mathtt{b2}\}) > T^{(\phi)}(\{\mathtt{b1}\}) = T^{(\phi)}(\{\mathtt{b3}\})$$
$$\text{if } \alpha < 0 \ : \ T^{(\phi)}(\{\mathtt{b2}\}) < T^{(\phi)}(\{\mathtt{b1}\}) = T^{(\phi)}(\{\mathtt{b3}\})$$

となる。尚、一般に補正第 3 項以降の総和は、第 2 項より大きくならない事が示される。

ここで見て来たように、縮退の有る場合には、似た特徴値を持ち補正項だけ小さな違いの有るログ、イベントが多く現れる。従って、特徴値の大きさそれ自身には大きな意味が無く、むしろ、特徴値の似たログやイベント同士の値の大小関係が重要な意味を持って来る。

4 適用例

本章では、我々の示した特徴値の有効性を評価する為 GNU bash-2.02 をターゲットとして現在行っている実験から適用例を紹介する。不具合の情報は、bash-2.02 以降の CHANGES、及び、ファイルの差分を元にしている。

ここで例として挙げる不具合は、bash に組み込まれたコマンド printf に関する物で、このコマンドに format 文字列として"%L"の様に修飾子だけで変換指定文字が無い物を与えた時に、bash が異常終了してしまう物である。異常終了するログは、

```
printf "hoge%L" 10
```

を実行させて記録した。一方、比較対象として記録した正常なログは、

```
printf "hoge%Ld" 10
```

の様なコマンドを実行させた物を三つ用意した。そして、比較起点として、main() 先頭を選び、4 つのログを切り出して比較を行った。

特徴値の計算の結果は、main() 直下に四ヶ所の疑わしい関数が有り、

- 浅い位置に有るイベント優先:
  - shell.c:set_shell_name()
  - shell.c:shell_initialize()
  - shell.c:maybe_make_restricted()
  - eval.c:reader_loop()
- 深い位置に有るイベント優先:
  - eval.c:reader_loop()

がそれぞれ最も大きな特徴値を持つ関数となる。これらの様子を図 9 に示した。関数 reader_loop() だけは、浅い位置のイベントを優先しても深い位置のイベントを優先しても疑わしいので、まず、この関数を調査する。この関数の呼び出し以下、疑わしい構造が一つだけと言う状況がしばらく続くので、それらを単純に辿って行くと、関数 printf.def:printf_builtin() に行き着く。この関数内には二ヶ所疑わしい部分が有る。

- 浅い位置に有るイベント優先:
  - printf.def:300 行目の default ラベルのブロック
- 深い位置に有るイベント優先:
  - printf.def:141 行目の for ループ

まず、浅い位置に有るイベント優先で示されるソースを調べると、これは、丁度不具合が起きた為に、エラーメッセージを表示して bash を異常終了させる部分である事が分かる。つまり不具合の観測される位

置その物であり、実際にはこれ以前に不具合が生じているはずである。

一方、深い位置に有るイベント優先で示される for ループには更に構造が有る。これ以下は単調に構造を辿る事が出来、最終的に printf.def:155 行目の分岐文 if (*fmt == '%') に到着する。我々のツールでは、これ以上の絞り込みは現在のところ行えない (比較起点を main() から変えれば、更に絞り込める可能性は有る)。一方、実際の不具合の原因は、この分岐で扱っている変数値に対する場合分けの不足であり、bash-2.03 では、printf.def:193 行目に新しい条件を付け加える事によって修正を行っている。従って、我々のツールは十分良い精度で不具合の原因の領域を指し示していると言う事が出来る。

特に、上で紹介した事例では、ユーザはソースコードに関する知識や、実行中の変数値等に関する知識を一切利用しないで、絞り込みを行える事に注意されたい。実際のデバッグ工程では、この絞り込みを行った後、デバッガ等を利用して、更に不具合の原因を調査する事になる。従来手法のみでデバッグしたならば、main() 関数先頭から調査する事によって辿り着く迄に掛かったであろう時間が、本手法では大幅に削減出来ている事が、この例から見て取る事が出来る。

5 まとめと今後の課題

本稿は、探索型デバッグ手法において特徴値の概念の占める役割に重点を置いて解説を行った。ここで紹介した特徴値の定め方は、従来の類似度の定義 [3] を流用した定め方を見直した物である。新しい定め方の有効性は現在実験中である。又、従来の定め方に比べ、どの程度有効になったの評価は、今後確認する予定である。

我々は、以前、従来の特徴値の定義を使ったツールで有効性の評価実験 [2] を行った。この結果は、十分、肯定的な物であった。しかし、従来の定義には、イベントの発生頻度だけでなく、関数等の構造の直下にどれだけ多くの部分構造が含まれているかと言う、静的情報が特徴値に大きく寄与してしまう問題が有った。本稿で示した特徴値の定義はこの問題を解決した物である。従って、我々は、これを用いる事により、更に的確にログ中の特徴的な振舞いを見出す事が出来る様になると考えている。

但し、今後の大きな課題として、深さの補正を加えて求めた特徴値を、ユーザにどの様にして提示するとツールとして使いやすくなるか、と言う問題が挙げられる。単純に、浅い所に有るイベントを優先して求めた疑わしい位置と、深い所に有るイベントを優先して求めた位置とを共に提示すると、むしろユーザはどちらから調べれば良いかとまどうかも知れない。この問題に対する回答は今後ツールをインプリメントし、被験者に使って貰いつつ探って行く予定である。

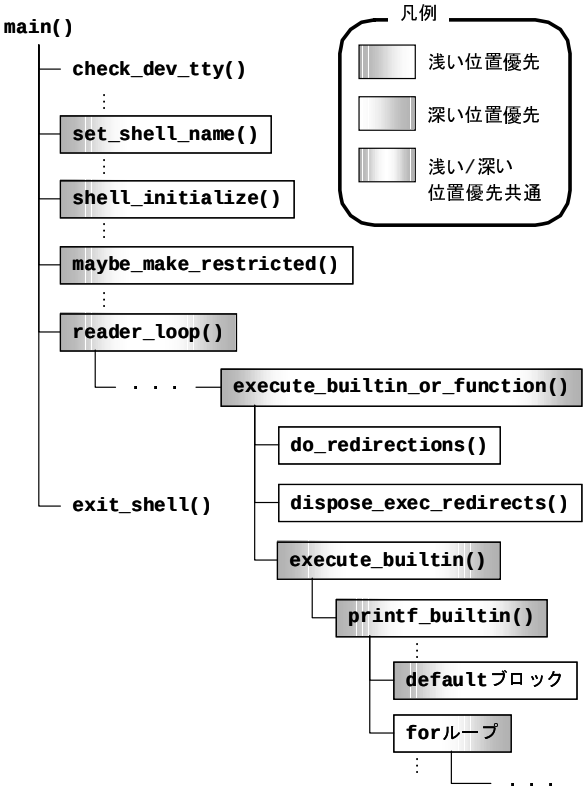

図 9: 適用例

参考文献

[1] 植木 克彦他, “探針型デバッグ手法の提案”, 信学技報 Vol.100, No.186, pp.1-8 (2000-7)

[2] 岡本 渉他, “探針型デバッグ手法の実現 (1) – 概要と評価”, 62 回情処学会全国大会 2Z-1 (2001-03)

[3] 田村 文隆他, “探針型デバッグ手法の実現 (2) – アルゴリズム”, 62 回情処学会全国大会 2Z-2 (2001-03)

[4] 植木 克彦他, “プログラム動作の類似度評価に基づく探索型デバッグ手法の拡張”, 第 43 回プログラミングシンポジウム pp101-108 (2002-1)

[5] 原田昌紀: サーチエンジンにおける検索結果のランキング, bit, Vol32, No.8, pp.8-14(2000)