特徴抽出と抽象化による動的バースマークの構成とその検証

M2004MM006 林 晃一郎
指導教員　青山 幹雄

1　はじめに

プログラムの盗用の発見を目的とする動的バースマークを, 実行時情報からの特徴の抽出とその抽象化の結果とする枠組みが提案されている [3]. この枠組みに従うことで, 盗用の発見に効果的な種々の動的バースマークが構成可能となる有効性が期待されるが, 十分な実例によるその検証はなされていない. 本研究ではこの検証を目的として, 抽象化操作の例をいくつか示し, それらに基づく動的バースマークを構成して実験・評価することを試みた. 実験の結果, 実際に期待される有効性を得た.

2　盗用と動的バースマーク

本節では, 本研究で想定するプログラムの盗用と, その発見技術である動的バースマークについて述べる.

2.1　プログラムの盗用

プログラムは盗用時に盗用の事実の隠蔽のため, 最適化や難読化 [5] などのプログラムの振舞いに影響を与えない変換が施される場合がある. この点を考慮し, 盗用を次に定義する[*1] .

定義 1 (盗用) プログラム P と P' がある. P' が P の完全な複製, または P の複製に変換を施したものならば, P' を P の盗用と言う.

2.2　動的バースマーク

動的バースマークとは [1–3] によると次に定義される.

定義 2 (動的バースマーク) $f(P,i)$ は入力 i によるプログラム P の実行時情報から, ある方法 f によって抽出された特徴群とする. Q をあるプログラムとして次の 2 つの性質を十分に満たすならば, $f(P,i)$ を P と i の動的バースマークと呼ぶ.

性質 1 (保存性) Q が P の盗用 $\Rightarrow f(P,i) \simeq f(Q,i)$.

性質 2 (弁別性) P と Q は互いに独立に実装されている $\Rightarrow f(P,i) \not\simeq f(Q,i)$.

つまり, 動的バースマークは変換されにくいプログラムそのものの特徴で, その一致により盗用の発見を可能とする. また, 電子透かし [5] のように予め情報を埋込む作業が必要なく, 配布済みプログラムにも有効である.

3　動的バースマーク構成の枠組みと関連研究

本節では動的バースマークの構成の枠組みを説明する.

3.1　特徴抽出と抽象化による動的バースマーク構成の枠組み

まず, 枠組みの説明のために以下の定義を用意する.

[*1] 人の手作業による改変は労力を要すため, 本研究ではツールによる機械的な変換が主流と考える.

定義 3 (実行系列) 系列 $T = trace(P,i)$ は入力 i に対するプログラム P の実行過程の全情報とし, $T = \langle t_1, \cdots, t_m \rangle$ と表す. これを P と i の実行系列と呼ぶ.

定義 4 (抽出系列) 系列 $S = \langle s_1, \cdots, s_m \rangle$, 着目する特徴を X とし, S からの X の抽出系列 $S/X = \langle s_{i_1}, \cdots, s_{i_n} \rangle$ を次の通り定める.

- $1 \leq i_1 < \cdots < i_n \leq m$
- $\{s_p\}$ の中で X の特徴を持つものは, $\{s_{i_q}\}$ と一致

P をプログラム, i を入力, X をある特徴, $abs(S)$ はある方法 abs で系列 S を抽象化した結果とすると, [3] は次の手順での動的バースマークの構成を提案している.

手順 1. 抽出系列 $E = trace(P,i)/X$ を求める

手順 2. 抽象表現 $f(P,i) = abs(E)$ を求める

つまり, **定義 2** の f を特徴の抽出とその抽象化という 2 つの操作の合成とする. これにより, 1 つの実行系列に対して, 複数の抽出系列の求め方 (X の選び方) と複数の抽象化方法 (abs) を互いに独立に選択し組み合わすことで, 種々の動的バースマークが構成できると期待されている (以降, 枠組みの有効性と呼ぶ).

3.2　関連研究との違い

[3] と従来の研究 [1,2] の違いを述べる. Myles ら [1] はプログラムコードの各基本ブロックの実行パターンのグラフ表現を, 岡本ら [2] は Windows OS 上のプログラムを対象として実行時の各 Windows API 呼出しの順序と頻度を動的バースマークとする. しかし, これらは古田ら [3] のように種々の動的バースマークの構成は述べていない. 動的バースマークは電子透かしのように著作者を識別する情報を用いない性質上, 異なるプログラム間で偶然に一致する可能性がある. そのため, 1 つだけではなく複数の動的バースマークの一致による盗用の発見が求められる. 古田らはこれに応えており, 変換で同時にいくつもの動的バースマークが変更されることは少ないと考えられ, 盗用の発見の信頼性を高められる.

4　動的バースマーク構成のための抽象化方法

3 節で動的バースマークの構成方法の枠組みを述べたが, [3] では抽象化操作の例を 1 つしか示していない. そのため, 現時点では抽出する特徴と抽象化方法の組合せはほとんどできず, 枠組みの有効性の検証もなされていない. 本節ではこの解決のために, 新たに抽象化方法を提案する. まず, 4.1 節で [3] の抽象化方法を紹介する. そして 4.2 節以降で, 実装者によるプログラムの処理の記述方法の違いに着目した抽象化方法を提案する. なお, 多数の逆コンパイラや難読化ツールの存在から, 盗用の事実を隠蔽されやすい言語の典型である Java を対象として話を進める.

4.1 繰返しパターンによる抽象化方法

プログラムの実行時情報には繰返し文などから繰返し現れる特徴があり, この実行時の繰返しをなくす変換は一般に困難である[*2] . [3] ではこの点から実行系列中の特徴の繰返しパターンによる抽象化方法を提案している.

a. **抽象化方法**

具体例で述べる. $trace(P,i)/X = \langle a,b,c,a,b,c,d,e,d,e,a,b,c,a,b,c,d,e,d,e\rangle$ を得たとする. メタ記号 (,), ^ を (,) はグループ化, ^n は直前の要素の n 回の繰返しとすると, 上記の系列は ((a,b,c)^2(d,e)^2)^2 と表せる. さらに, 繰返し内の細部の処理 (ここでは a,b,c,d,e) は変換されやすい[*3] ため除去し, (()^2()^2)^2 という抽象表現を動的バースマークとする.

b. **特徴 X の例**

$trace(P,i)/X$ の特徴 X は例えばメソッド呼出しやフィールド変数値の変更を選択できる. 実際に [3] ではこれらの特徴のそれぞれから保存性と弁別性を十分に満たす動的バースマークが構成できている. 従来ではこれらの特徴は変換されやすいとされ, あまり用いられなかった. 一方で, 上述の抽象化では繰返し内の変換に弱い箇所を除去するため, [3] はこれらの特徴も抽象化の工夫で選択できることを示したと言える.

4.2 出現間隔による抽象化方法

実装者により実行時情報中の各特徴の位置は異なる. そこで, これらの各位置の間隔による抽象化を提案する.

a. **抽象化方法**

その抽象化による動的バースマークを**定義 6** とする.

定義 5 (出現間隔列) 系列 S からの特徴 X の抽出系列を $S/X = \langle s_{i_1},\cdots,s_{i_n}\rangle$ とする. S による X の出現間隔列 $interval_S(X)$ を次で定める.

$$interval_S(X) = \langle i_2 - i_1,\cdots,i_n - i_{n-1}\rangle$$

定義 6 (出現間隔バースマーク) P をプログラム, i を入力, 各 $X,Y_1,\cdots,Y_p$ をある特徴, $E = trace(P,i)/X$, $I_j = interval_E(Y_j)$ $(1 \le j \le p)$ とおく. 順序組 $(I_1,\cdots,I_p)$ を E による $Y_1,\cdots,Y_p$ の出現間隔バースマークと呼ぶ.

このように**定義 6** は各特徴の間隔を表したものである. 実行系列の一部のみの変換では, これらの間隔の一部しか変更できないため, 少々の変換には出現間隔バースマークは耐性があると期待できる.

b. **特徴 X の例**

定義 6 の特徴 $Y_1,\cdots,Y_p$ が変換されやすいと, 各出現間隔列 I_j も形態が変更されやすくなる. そのため, $Y_1,\cdots,Y_p$ は変換に強い特徴にした方がよい. Java ではこのような特徴は API メソッド名や API クラス名に相当する. なぜなら, API 関連の変換はそのライブラリの解析や変換も生じ, 非常に困難だからである[*4] . よって, 抽出系列 E がこれらの特徴を含むように X をメソッド呼出しとしたり, API クラス (`java.lang.String`, `java.util.Vector` など) のオブジェクト (以降, API オブジェクトと呼ぶ) も変換が困難だから, X にオブジェクトへの参照を選択できる.

c. **動的バースマークの比較方法**

P と P' をプログラム, i を入力, 各 $X,Y_1,\cdots,Y_p$ をある特徴, $E = trace(P,i)/X$, $E' = trace(P',i)/X$, $I_j = interval_E(Y_j)$, $I'_j = interval_{E'}(Y_j)$ $(1 \le j \le p)$ とし, 出現間隔バースマーク $(I_1,\cdots,I_p)$ と $(I'_1,\cdots,I'_p)$ があるとする. この 2 つの比較には以下の**定義 7** を用い, $R_j = SR(I_j,I'_j)$ として $R_1,\cdots,R_p$ の多くが 1 の数値に近い場合, P と P' は盗用の関係にある可能性が高いとする.

定義 7 (2 系列の一致率) 2 系列 $A = \langle a_1,\cdots,a_m\rangle$, $B = \langle b_1,\cdots,b_n\rangle$ の最長共通部分列の長さ $L_{A,B}(m,n)$ (以降, 添字 A,B は省略) は次で与えられるとする.

$$L(m,n) = \begin{cases} L(m-1,n-1)+1 & (a_m = b_n) \\ \max\{L(m,n-1),L(m-1,n)\} & (a_m \ne b_n) \end{cases}$$

2 系列 A と B の一致率 $SR(A,B)$ を次で定める.

$$SR(A,B) = \frac{2\cdot L(m,n)}{m+n}$$

4.3 近傍集合による抽象化方法

互いに近い時点で実行される処理同士は関連がある場合が多い. そこで, 実行系列中に互いに近い位置に現れる特徴同士の関係を集合として表す抽象化を提案する.

a. **抽象化方法**

その抽象化による動的バースマークを**定義 9** とする.

定義 8 (近傍集合) 系列 $S = \langle s_1,\cdots,s_m\rangle$, 各 X, Y_1, $\cdots,Y_p$ をある特徴, $S/X = \langle s_{i_1},\cdots,s_{i_n}\rangle$ とする. $g(s_u)$ $(1 \le u \le m)$ は要素 s_u が $Y_1,\cdots,Y_p$ のいずれかの特徴を持つならば s_u, そうでなければ空を表し, d (≥ 0) を定数とする. S, 範囲 d, $Y_1,\cdots,Y_p$ による X の近傍集合 $N[Y_1,\cdots,Y_p]^d_S(X)$ を次で定める.

$$N[Y_1,\cdots,Y_p]^d_S(X) = \{g(s_u) : 1 \le \exists u \le m, 1 \le \exists v \le n, |u - i_v| \le d\}$$

定義 9 (近傍集合バースマーク) P をプログラム, i を入力, 各 $X,Y_1,\cdots,Y_p$ をある特徴, d (≥ 0) を定数とし, $E = trace(P,i)/X$, $S_j = N[Y_1,\cdots,Y_p]^d_E(Y_j)$ $(1 \le j \le p)$ とおく. 順序組 $(S_1,\cdots,S_p)$ を E と範囲 d による $Y_1,\cdots,Y_p$ の近傍集合バースマークと呼ぶ.

このように**定義 9** では集合としての抽象化によって各要素の出現順序の情報は捨象する. これにより各処理の実行順序が多少変更されても動的バースマークの変化は少ないことが期待できる.

b. **特徴 X の例**

定義 9 における特徴 $Y_1,\cdots,Y_p$ が変換されやすいと各近傍集合 S_j も形態が変更されやすくなるから, これらは API 関連の特徴とし, X は**定義 6** と同様にメソッド呼出しやオブジェクトへの参照にするとよい.

c. **動的バースマークの比較方法**

近傍集合バースマーク同士の比較には以下の**定義 10**

[*2] 繰返し文に対する難読化もあるが, if 文への展開などの表現上の変換にとどまり実行時情報を変化させないものが大半である.

[*3] メソッドのインライン化, 変数名の変更などの難読化がある.

[*4] 一方で, ユーザ定義のメソッドとクラスに対しては種々の難読化が提案されており, 名前の変更も容易である.

を用いる．この定義において $NR(B,B')$ の数値が 1 に近い場合，プログラム P と P' は盗用の関係にある可能性が高いとする．

定義 10 (近傍集合バースマークの一致率) P と P' をプログラム，i を入力，$X,Y_1,\cdots,Y_p$ のそれぞれをある特徴，d (≥ 0) を定数とし，$E = trace(P,i)/X$，$E' = trace(P',i)/X$，$S_j = N[Y_1,\cdots,Y_p]^d_E(Y_j)$，$S'_j = N[Y_1,\cdots,Y_p]^d_{E'}(Y_j)$ $(1 \leq j \leq p)$ とおく．2 つの近傍集合バースマーク $B = (S_1,\cdots,S_p)$，$B' = (S'_1,\cdots,S'_p)$ の一致率 $NR(B,B')$ を次で定める．

$$NR(B,B') = \frac{2\sum_{j=1}^{p}|S_j \cap S'_j|}{\sum_{j=1}^{p}(|S_j| + |S'_j|)}$$

4.4 スライドバッファによる抽象化方法

プログラムの各処理はある時点で頻繁に実行されるなどのばらつきがあり，これは実装者によって異なるであろう．そこで，実行系列中を長さ k (k は定数) の範囲 (以降，*k-gram* バッファと呼ぶ) で調べて，その範囲内に含まれる各特徴の個数を用いた抽象化を提案する．

a. 抽象化方法

その抽象化による動的バースマークを**定義 13** とする．

定義 11 (*k-gram* 個数) 系列 $S = \langle s_1,\cdots,s_m\rangle$，$X$ をある特徴とし，$S/X = \langle s_{i_1},\cdots,s_{i_n}\rangle$ とする．$k \geq 0, 1 \leq r \leq m-k$ とし，"$(i_{p-1} < r \leq i_p) \wedge (i_q \leq r+k < i_{q+1})$" を満たす p,q があるとき，S と区間 $[r,r+k]$ による X の *k-gram* 個数を $num^{r,k}_S(X) = q-p+1$ で定める．

定義 12 (*k-gram* 個数列) 系列 $S = \langle s_1,\cdots,s_m\rangle$，ある特徴を X とし，$K_r = num^{r,k}_S(X)$ $(k \geq 0, 1 \leq r \leq m-k)$ とおく．S と範囲 k による X の *k-gram* 個数列 $seqnum^k_S(X)$ を次で定める．

$$seqnum^k_S(X) = \langle K_1,\cdots,K_{m-k}\rangle$$

定義 13 (スライドバッファバースマーク) P をプログラム，i を入力，各 $X,Y_1,\cdots,Y_p$ をある特徴，k (≥ 0) を定数，$E = trace(P,i)/X$，$B_j = seqnum^k_E(Y_j)$ $(1 \leq j \leq p)$ とする．順序組 $(B_1,\cdots,B_p)$ を E と長さ k による $Y_1,\cdots,Y_p$ のスライドバッファバースマークと呼ぶ．

このように**定義 12** では，r の値を 1 から $m-k$ まで変化させて，つまり *k-gram* バッファを系列上でシフトさせ，逐次バッファ中の特徴の個数を列挙した系列を定義している．**定義 13** はこの系列を用い，E 中の各特徴のばらつきを表現した動的バースマークと言える．これも**定義 9** と同様に特徴の出現順序の情報は捨象している．

b. 特徴 X の例

定義 13 における特徴 $Y_1,\cdots,Y_p$ が変換されやすいと各 *k-gram* 個数列 B_j も形態が変更されやすくなる．そのため，**定義 6** と**定義 9** と同様に $Y_1,\cdots,Y_p$ は API 関連の特徴とし，X はメソッド呼出しやオブジェクトへの参照にするとよい．

c. 動的バースマークの比較方法

P と P' をプログラム，i を入力，各 $X,Y_1,\cdots,Y_p$ をある特徴，$E = trace(P,i)/X, E' = trace(P',i)/X, B_j = seqnum^k_E(Y_j), B'_j = seqnum^k_{E'}(Y_j)$ $(1 \leq j \leq p)$ とし，スライドバッファバースマーク $(B_1,\cdots,B_p)$ と $(B'_1,\cdots,B'_p)$ があるとする．この 2 つの比較には**定義 7** を用い，$Q_j = SR(B_j,B'_j)$ として $Q_1,\cdots,Q_p$ の多くが 1 の数値に近い場合，P と P' は盗用の関係にある可能性が高いとする．

5 評価実験

本節では 4 節で提案した抽象化方法に基づく動的バースマークを構成し，それらの評価実験について述べる．

5.1 評価方法

動的バースマークの評価は，保存性と弁別性の確認で行う．本研究では保存性の確認に種々の難読化方法を提供するツールである SandMark [6] を用い，対象とするプログラムの難読化前と難読化後の動的バースマークの一致を調べた．この場合，一致があるほど保存性を満たすと言える．弁別性の確認には，異なる実装者による仕様の等しいプログラムをいくつか用意し，それらから構成される動的バースマーク同士の一致を調べた．この場合，一致が小さいほど弁別性を満たすと言える．

5.2 保存性の確認結果

jdepend-2.9.1.jar [7], java2html.jar [8], tar.jar [9] の 3 個のツールプログラムを対象とし，それぞれから**定義 6**, **定義 9**, **定義 13** の動的バースマークの難読化の耐性を確認した．その結果，これらのプログラムの全てから同様の傾向を得た．本節では，このうち jdepend-2.9.1.jar を用いた結果を述べる．なお，以降，jdepend-2.9.1.jar を P, P を難読化したものを P' とおく．

5.2.1 出現間隔バースマークの耐性

定義 6 における X を API メソッド呼出し，各 $Y_1,\cdots,Y_p$ をある 15 個の API メソッド名 $(p = 15)$ とし，P と P' から構成される出現間隔バースマーク同士を比較した．その結果を $E = trace(P,i)/X$，$E' = trace(P',i)/X$，$I_j = interval_E(Y_j)$，$I'_j = interval_{E'}(Y_j)$ とし，各 j $(1 \leq j \leq p)$ に対し $SR(I_j,I'_j) \geq 0.8$ となる個数が n 個の場合，n/p の値 (小数点第 3 位を四捨五入) を**表 1** の method の列に示す．なお，表の難読化方法は SandMark が提供する難読化の種類である．また，同様に X を API オブジェクトを値として保持するフィールド変数値の参照，各 $Y_1,\cdots,Y_p$ をある 7 個の API オブジェクトへの参照 $(p = 7)$ として比較した結果を**表 1** の field の列に示す．

表からいくつかの難読化に耐性が低い．Array Splitter は API メソッドを用いて動的にフィールドの配列を生成し追加する．Class Splitter は各クラス間の継承関係を複雑にし，各フィールド変数値の参照にオーバヘッドを生む．Promote Primitive Types はプリミティブ型をラッパクラスの型に変換し，この型の操作には API メソッドが使われる．つまり，これらはメソッド呼出しやフィールド変数値の参照を余分に追加するため耐性が低くなると考えられる．一方で他の難読化には耐性がある．

5.2.2 近傍集合バースマークの耐性

定義 9 における X をメソッド呼出し，Y_1, $\cdots$, Y_p を

表 1　出現間隔バースマークの耐性

難読化方法	method	field
Array Splitter	0.33	0.43
Class Splitter	1.00	0.00
Dynamic Inliner	1.00	1.00
Inliner	1.00	1.00
Method Merger	1.00	1.00
Objectify	1.00	1.00
Overload Names	1.00	1.00
Promote Primitive Types	0.00	1.00
Publicize Fields	1.00	1.00
Split Classes	1.00	1.00
Static Method Bodies	1.00	1.00

表 2　近傍集合バースマークの耐性

難読化方法	method			field		
	2	4	6	2	4	6
Array Splitter	0.89	0.90	0.89	0.91	0.90	0.90
Class Splitter	1.00	1.00	1.00	0.97	0.85	0.89
Dynamic Inliner	0.90	0.85	0.82	1.00	1.00	1.00
Inliner	0.98	0.99	0.99	1.00	1.00	1.00
Method Merger	1.00	1.00	1.00	1.00	1.00	1.00
Objectify	1.00	1.00	1.00	1.00	1.00	1.00
Overload Names	1.00	1.00	1.00	1.00	1.00	1.00
Promote Primitive Types	0.90	0.93	0.91	1.00	1.00	1.00
Publicize Fields	1.00	1.00	1.00	1.00	1.00	1.00
Split Classes	1.00	1.00	1.00	0.00	0.00	0.00
Static Method Bodies	0.83	0.78	0.78	1.00	1.00	1.00

表 3　スライドバッファバースマークの耐性

難読化方法	method			field		
	2	4	6	2	4	6
Array Splitter	0.40	0.40	0.40	0.43	0.43	0.43
Class Splitter	1.00	1.00	1.00	0.00	0.00	0.00
Dynamic Inliner	1.00	1.00	1.00	1.00	1.00	1.00
Inliner	1.00	1.00	1.00	1.00	1.00	1.00
Method Merger	1.00	1.00	1.00	1.00	1.00	1.00
Objectify	1.00	1.00	1.00	1.00	1.00	1.00
Overload Names	1.00	1.00	1.00	1.00	1.00	1.00
Promote Primitive Types	0.07	0.27	0.20	1.00	1.00	1.00
Publicize Fields	1.00	1.00	1.00	1.00	1.00	1.00
Split Classes	1.00	1.00	1.00	1.00	1.00	1.00
Static Method Bodies	1.00	1.00	1.00	1.00	1.00	1.00

$E = trace(P, i)/X$ 中に含まれる全ての API メソッド名とし, P と P' から 2 つの近傍集合バースマークを構成し比較した. その結果を**定義 10** の尺度による値 (小数点第 3 位を四捨五入) で**表 2** の method の列に示す. なお, **定義 9** における d の値を 2, 4, 6 とした場合をさらに列に分けて示す. また, X をフィールド変数値の参照, $Y_1, \cdots, Y_p$ を E 中に含まれる全ての API オブジェクトへの参照とし, 同様に P と P' から構成される動的バースマークを比較した結果を**表 2** の field の列に示す.

表の結果から Split Classes の難読化に耐性が低い. この難読化は各クラスを 2 分割し, 分割したクラスのオブジェクトを新たに追加したフィールド変数に保持させて扱う. つまり, 実行時のフィールド変数値の参照を増やすため, 表の field の列の数値が低くなったと考えられる. 一方で他の難読化には高い耐性を示している.

5.2.3　スライドバッファバースマークの耐性

定義 13 における $X, Y_1, \cdots, Y_p$ を 5.2.1 節と同じにし, P と P' から 2 つのスライドバッファバースマークを構成し比較した. この比較も 5.2.1 節と同様に**定義 7** を用い, p 個中 n 個の 2 系列の一致率が 0.8 以上の数値の場合, n/p の値で結果を**表 3** に示す. なお, **定義 13** における k の値を 2, 4, 6 とした場合をさらに列に分けて示す. 5.2.1 節と同様の理由で耐性の低い難読化があるが, ほとんどに対し耐性を示していると言える.

5.3　弁別性の確認結果

弁別性の確認は, 仕様は等しいが異なる 3 人によって実装された 3 つのプログラム間で動的バースマークを比較して行った. 各動的バースマークは**定義 6**, **定義 9**, **定義 13** における $X, Y_1, \cdots, Y_p$ を 5.2.1 節, 5.2.2 節, 5.2.3 節のそれぞれのメソッド呼出しおよびいくつかのメソッド名を選択し構成したもので, 比較の結果, これらの動的バースマークはほとんど一致しなかった.

6　実験結果の考察と今後の課題

5 節の実験から枠組みの有効性を考察する. 5.2 節で示した各表から耐性を示さない難読化があったが, これらの大半は一方の特徴を用いたものは耐性が低いが, もう一方は高い耐性を示している. つまり, 複数の特徴の抽出と複数の抽象化方法によって種々の動的バースマークを構成でき, 互いに弱点を補完させられる. また, ほとんどの難読化に対し十分に耐性を有し, 弁別性も示された. 以上から枠組みの有効性が得られたと言える.

今後の課題としては, より多数の動的バースマークを構成できるように実行時情報から種々の特徴を抽出する方法や, その他の抽象化方法の検討が必要である.

7　まとめ

本論文では実行時情報からの特徴の抽出とその抽象化による動的バースマークの構成の枠組みを扱い, 抽象化の例として, 出現間隔バースマーク, 近傍集合バースマーク, スライドバッファバースマークを提案した. そして, 実行時情報からメソッド呼出しとフィールド変数値の参照の特徴を抽出し, 各抽象化方法を用いて動的バースマークを構成し, 評価した. その結果, 枠組みに従うことで種々の有効な動的バースマークを構成できた.

8　謝辞

研究の助言を頂いた南山大学数理情報学部の真野芳久教授ならびに青山幹雄教授に心から感謝致します.

参考文献

[1] G. Myles, *et al.*: “Detecting Software Theft via Whole Program Path Birthmarks”, *Proc. the 7th Inf. Security Conf.*, Sep. 2004, pp. 404-415.

[2] 岡本, 他: “ソフトウェア実行時の API 呼び出し履歴に基づく動的バースマークの実験的評価”, 情処学会 第 46 回プログラミング・シンポジウム報告集, Jan. 2005, pp. 41-50.

[3] 古田, 他: “実行系列の抽象表現を利用した動的バースマーク”, 信学論 D-I, Vol. J88-D1, No. 10, Oct. 2005, pp. 1595-1598.

[4] 林, 他: “特徴抽出と抽象化による動的バースマークの構成とその検証”, 情処研報, Vol. 2005, No. 122, 2005-CSEC-31, Dec. 2005, pp. 31-36.

[5] C. S. Collberg, *et al.*: “Watermarking, Tamper-Proofing, and Obfuscation – Tools for Software Protection”, *IEEE Trans. on Softw. Eng.*, Vol. 28, No. 8, Aug. 2002, pp. 735-746.

[6] “SandMark”, http://sandmark.cs.arizona.edu/.

[7] “JDepend”, http://www.clarkware.com/software/JDepend.html.

[8] “Java2Html”, http://www.java2html.de/.

[9] “Java Tar”, http://www.trustice.com/java/tar/.