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

林 晃一郎[†,☆] 楓 基靖[†] 真野芳久[††]

我々は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，プログラムの実行時情報からの特徴の抽出とその特徴を簡潔に表現する抽象化の 2 つの操作の組合せによって構成する枠組みを提案している．この枠組みに従い，2 つの操作のそれぞれを独立に選択して組み合わせることで，1 つの実行時情報から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になるという有効性が期待できる．実際に我々は枠組みに従って動的バースマークをいくつか構成し，それらを実験で評価することでこの有効性の検証を行った．その結果，期待される有効性を確認できた．

Feature-abstraction Framework to Construct Dynamic Birthmarks and some Experiments

Koichiro Hayashi,[†,☆] Motoyasu Kaede[†] and Yoshihisa Mano[††]

We propose a framework to construct dynamic birthmarks for detecting the program thefts. In this framework, birthmarks are constructed by combining two operations, the extraction of some feature from run-time information, and the abstraction of the data with the feature. Since this framework makes us consider each operation individually, we can expect gaining various effective birthmarks from one run-time information. Actually, we have constructed some birthmarks using this framework, and demonstrated its effectiveness.

1. はじめに

プログラムの知的財産権を侵す例として，プログラムそのものをまたは何らかの細工を施したものをあたかも独自開発したかのように偽装して世に出すことがある．この種のプログラムの盗用の発見を目的とした技術に動的バースマークがある[1),2)]．動的バースマークとは，制御の流れやデータへのアクセスなどのプログラムの実行時の動作の内容（以降，実行時情報と略記）の特徴を何らかの形で表現したもの，またはその表現の一致をもってプログラムの盗用を発見する技術を指す．電子透かし[3)]のようにあらかじめ情報を埋め込む作業が必要でないため，動的バースマークは配布済みのプログラムにも適用できる技術である．一方で，著作者などを一意に識別する情報を用いない性質から，動的バースマークは異なるプログラム間で偶然に一致する可能性を持つ．このことから，動的バースマークは 1 つの実行時情報から複数構成して比較することが望ましく，また，そうすることで盗用の発見をより確実に行うことができる．しかし，従来の研究[1),2)]では，1 つの実行時情報から 1 つのみの動的バースマークを構成する方法を示すにとどまり，種々の動的バースマークを系統的に構成する方法については議論されていない．

この問題に対して我々は，動的バースマークを実行時情報からの特徴の抽出と，抽出された特徴の抽象化との 2 つの操作の組合せと考える枠組みを提案している[4)]．この枠組みでは，抽出する特徴（以降，抽出特徴と略記）の種類と抽象化の方法（以降，抽象化方法と略記）の種類とを互いに独立に選択して組み合わせることで，種々の動的バースマークが構成可能になるという有効性が期待できる（以降，これを枠組みの有効性と呼ぶ）．文献 4) ではこの枠組みの基となるアイデアを示しているが，その一例として 1 つのみの抽象化方法を提示したにとどまっている．すなわち，抽出特徴と抽象化の種々の組合せを実現しておらず，枠組みの有効性を検証していない．そこで，我々は新たに抽象化方法の例を提示し，様々な抽出特徴と抽象化方法を組み合わせて動的バースマークを構成する実験を

† 南山大学大学院数理情報研究科
Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
†† 南山大学数理情報学部
Faculty of Mathematical Sciences and Information Engineering, Nanzan University
☆ 現在，株式会社インテック
Presently with INTEC INC.

行い，この有効性の確認を試みた．本稿ではこの枠組みをより正確に述べ，有効性の検証の結果を報告する．

まず，2 章では我々が想定するプログラムの盗用と動的バースマークの定義を示す．3 章では提案する枠組みについて述べる．4 章では枠組みに従って動的バースマークを構成する際に行う抽象化方法の例をいくつか示す．5 章では 4 章で示した抽象化方法を用いて，枠組みに従って動的バースマークの構成を行った実験について述べる．また，構成された動的バースマークが盗用の発見に効果的であることを評価した結果を示す．6 章では 5 章で示された評価結果を基に，枠組みの有効性が実際に得られることの確認を行う．7 章では本稿のまとめと本研究の今後の課題を述べる．なお，本稿では多数の逆コンパイラや難読化[3] ツールの存在から，特に盗用が危険視される Java プログラムを動的バースマークの構成対象として話を進める☆．

2. プログラムの盗用と動的バースマーク

本章では，本研究で想定するプログラムの盗用と動的バースマークの定義を示す．

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，仕様に影響を与えない範囲でプログラムコードの外見や動作が機械的に変更される場合がある．そのような変更には，最適化や難読化などのプログラム変換の手法が一般的によく用いられる．この点を考慮し，盗用を次に定義する☆☆．

定義 1（盗用） プログラム P と P' がある．次の 2 つの条件の少なくとも 1 つを満たすならば，P' を P の盗用といい，この事実を $P' <\equiv_{theft} P$，この事実が成り立たないことを $P' <\not\equiv_{theft} P$ と表す．

- P' は P の複製である．
- P' は P の複製に変換を施したものである．

$P <\equiv_{theft} P'$ または $P' <\equiv_{theft} P$ の事実が成り立つ場合，P と P' は盗用の関係にあるという．

2.2 動的バースマーク

動的バースマークは従来の研究[1),2)] によると次に定義される．なお，2 つのある表現 E と E' があるとし，この 2 つを何らかの方法で比較した結果，互いに十分に一致する場合を $E \fallingdotseq E'$，一致しない場合を $E \not\fallingdotseq E'$ と表すことにする．

定義 2（動的バースマーク） $f(P,i)$ はプログラム P に入力 i を与えたときの実行時情報から，ある方法 f によって得られる何らかの表現とする．あるプログラム Q があって次の 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） $P <\equiv_{theft} Q$ または $Q <\equiv_{theft} P$ $\Rightarrow f(P,i) \fallingdotseq f(Q,i)$．

性質 2（弁別性） $P <\not\equiv_{theft} Q$ かつ $Q <\not\equiv_{theft} P$ $\Rightarrow f(P,i) \not\fallingdotseq f(Q,i)$．

保存性はプログラムに変換が施されていても構成される動的バースマークはほぼ同一であること，すなわち変換に耐性があることを意味する．弁別性は，同一仕様のプログラムであっても，実装者の実装方法の違いが動的バースマークに反映されるることを意味する．すなわち，動的バースマークは変換されにくい実行時情報の特徴を表し，その一致により盗用の発見を可能とする．また，電子透かしのようにあらかじめ情報を埋め込む作業が必要でないため，配布済みのプログラムにも有効である．

以降では，動的バースマークの構成の枠組みとそれに基づいて得られる動的バースマークの候補について検討を進めるが，記述を簡潔にするため，定義 2 を満たすことが期待される動的バースマークの候補を単に動的バースマークと記す．

3. 動的バースマーク構成の枠組みと関連研究

動的バースマークは，定義 2 における方法 f を複数考案することで，1 つの実行時情報からいくつも構成できる．しかし，保存性と弁別性を満たすように複数の異なる方法 f を発見する系統的な方法は，従来の研究[1),2)] において議論がなされていない．我々はこの点に対し，動的バースマークを系統的に構成するための枠組みを提案している[4)]．

3.1 動的バースマークの構成の枠組み

以下の定義を用意したうえで枠組みを詳しく述べる．

定義 3（実行系列） 系列 $T = trace(P,i)$ はプログラム P に入力 i を与えて実行したときの実行過程の全情報とし，時系列的に $T = \langle t_1, \cdots, t_m \rangle$ と表す．これを P の i による実行系列と呼ぶ．

定義 4（抽出系列） 系列 $S = \langle s_1, \cdots, s_m \rangle$，$X$ をある特徴とする．S から X の特徴を持つ要素だけを抽出した系列 $S/X = \langle s_{i_1}, \cdots, s_{i_n} \rangle$ を S からの X の抽出系列と呼び，次の 2 つを満たすとして定める．

(1) $1 \leq i_1 < \cdots < i_n \leq m$

(2) $\{s_j\}$ の中で特徴 X を持つものは $\{s_{i_k}\}$ と一致

P をプログラム，i を入力，X をある特徴，$abs(E)$ はある方法 abs による系列 E の抽象化の結果（抽象

☆ 従来においても Java プログラムを対象として，難読化や電子透かし，動的バースマークに関する研究がさかんに行われている．

☆☆ 人の手作業による改変は労力を要するため，従来から盗用者はツールによる機械的な変換を行うと一般に考えられている．

表現）とし，枠組みでは次の手順の結果を動的バースマークとして構成する．

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．

すなわち，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．

3.2 従来の関連研究と枠組みとの構成方法の違い

Myles ら[1] は Java バイトコードの各基本ブロックの実行パターンを文脈自由文法による生成規則で表し，その表現をグラフ化したものを，岡本ら[2] は Windows OS 上のプログラムを対象として実行時の Windows API 呼び出しの順序と頻度を動的バースマークとする．前者は基本ブロックを特徴として抽出してそれらの実行パターンをグラフで抽象化するが，これ以外の特徴と抽象化の組合せは考慮していない．後者は実行時の改変が困難な特徴☆を抽出して得た結果を動的バースマークとしている．両者とも盗用の発見に効果的な動的バースマークを構成しているといえるが，さらに別の種類の動的バースマークの構成には新たなアイデアを必要とし，系統的に種々の動的バースマークの構成を行うことについては議論していない．

動的バースマークを構成するためには，保存性と弁別性を持つように材料としての実行時情報を加工することになる．実行時情報の獲得には技術的問題あるいは時間的・空間的な問題を解決しなければならないことがある．問題が解決され従来は得られなかった実行時情報が獲得可能となったときに，それを材料とすることで新たな動的バースマークを構成できることが期待できる．また，加工方法は保存性や弁別性への影響が大きいと考えられるが，有望な加工方法を見出したときにこれまでに得られている動的バースマークで使われている実行時情報への適用によって新たな動的バースマークを構成できることが期待できる．

我々はこの考えを明確にするために，1 つの実行系列に対して複数の抽出系列の求め方（X の選択）と複数の抽象化方法（abs）の選択とを互いに別個に行い，これらを様々に組み合わせることを提唱している．そしてこの組合せから種々の動的バースマークが構成できることを本稿で示す．

動的バースマークはその性質上，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一般に求められる．また，盗用者側の対抗処置を考えると，利用可能な動的バースマークの集合は十分大きく新たに追加可能であることも求められる．我々の枠組みはこれに応えている．

4. 動的バースマーク構成のための抽象化方法

本章では，枠組みに従って動的バースマークを構成する際に行う抽象化の例をいくつか提示する．まず，4.1 節では，文献 4) に示されている唯一の抽象化方法を簡単に紹介し，4.2 節以降では新たに提案する抽象化方法を述べる．なお，これらの抽象化は，実行時情報から抽出した特徴群中で改変されにくい，または実装者ごとに異なるであろう情報（プログラムの記述方法などの違い）に着目し，これら以外の情報はできるだけ捨象するものである．そうすることで，保存性と弁別性を十分に満たす動的バースマークの構成が期待できる．本章では，P をプログラム，i を入力，X をある特徴として例などを記述する．

4.1 実行系列の繰返しパターンによる抽象化方法

本節では文献 4) にある抽象化方法を簡単に紹介する．

プログラムの実行時情報には繰返し文などから繰り返し現れる特徴がある．実行時の処理の繰返し回数や順序などの実行パターンの変更は，プログラムの仕様を変更する可能性が高いために一般に困難である☆☆．また，同じアルゴリズムを用いたとしても，メソッドを呼び出す位置，変数値を参照する位置などの実装者によるプログラムの記述の違いは実行時情報にも反映されるであろう．これら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考えることができる．

a. 抽象化方法

たとえば，$trace(P, i)/X = \langle a, b, c, a, b, c, d, e, d, e, a, b, c, a, b, c, d, e, d, e\rangle$ 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 $((a, b, c)\hat{}2(d, e)\hat{}2)\hat{}2$ と表せる．この表現に対し，繰返し内の具体的な処理内容（ここでは a，b，c，d，e）を捨て，(()^2()^2)^2 という抽象表現を動的バースマークとする．

b. 特徴の例

抽出系列 $trace(P, i)/X$ の特徴 X は，たとえばメソッド呼び出しやフィールド変数への値の代入の動作を選択できる．これらの 2 つの抽出特徴に対し，この抽象化を用いてそれぞれから動的バースマークを構成する実験結果の報告では，難読化を施したプログラム

☆ OS の API 関連の改変は OS 内部のライブラリにも改変が生じるため，一般に困難である．

☆☆ 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．

からも難読化前と同様の動的バースマークの構成ができたとされる[4]．従来ではこれらの特徴は変換されやすいとして動的バースマークの構成にあまり用いられなかったが，繰返し内の具体的な処理内容を捨てることによって保存性を満たす動的バースマークが構成できたといえる．このように抽象化の工夫で従来では用いられなかった特徴も利用できる場合がある．また，弁別性の確認もされている．

4.2 特徴の出現間隔を利用した抽象化方法

実装者によってプログラムの記述は異なる．この違いにより実行時情報中に現れる各処理の位置も異なると考えられる．この点に着目し，これらの各位置の間隔を利用した抽象化方法を提案する．

a. 抽象化方法

たとえば，$trace(P,i)/X = \langle a, x_1, b, c, d, x_2, x_3, e, f, x_4, g, x_5, \cdots \rangle$ を得たとする．ここで，$\{x_i\}_{i=1,2,\ldots}$ は $\{ a, b, c, \cdots \}$ とは区別可能な特徴を持つ要素として，これらの出現間隔の列（ここでは，$\langle 4, 1, 3, 2, \cdots \rangle$）を動的バースマークとするものである．次の定義で正確に述べる．

定義 5（出現間隔列） 系列 $S = \langle s_1, \cdots, s_m \rangle$ と特徴 Y があって $S/Y = \langle s_{i_1}, \cdots, s_{i_n} \rangle$ とする．S による Y の出現間隔列 $intv_S(Y)$ を次で定める．

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

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，$E = trace(P,i)/X$，$I_j = intv_E(Y_j)$ $(1 \le j \le p)$ 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．

例として $E = \langle x_1, \cdots, x_m \rangle$，$E/Y_j = \langle x_{j_1}, \cdots, x_{j_n} \rangle$ とし，定義 6 における I_j を図示すると**図 1** のようになる．この図は時間軸を意味する右矢印で系列を表し，特徴 Y_j を持つ要素の出現位置の間隔を列挙している様子を示したものである．このように定義 6 は各特徴の出現位置の間隔を表したものである．また，系列の一部分のみを変換してもこれらの複数の間隔のうちの一部しか変更できない．たとえば，図 1 のように系列 $\langle x_{j_1}, \cdots, x_{j_n} \rangle$ があって，要素 x_{j_2} がある位置から要素 x_{j_3} がある位置の間に余分な要素を挿入する変換がされたとする．この場合，x_{j_2} から x_{j_3} の間隔は変化するが，それ以外の x_{j_1} から x_{j_2} や $x_{j_{n-1}}$ から x_{j_n} などの間隔には影響がない．すなわち，少々の変換には出現間隔バースマークは保存性があると期待できる．なお，実行系列 $trace(P,i)$ ではなく抽出系列 E を基に $Y_1, \cdots, Y_p$ の特徴に着目して p 個の出現間隔列を求める理由は，実行系列はあらゆる特徴を含み，それらのいくつかは変換される可能性があるからである．実行系列が変換されると，求められる出現間隔列が変化してしまう可能性がある．そこで，着目したい特徴を X だけに限定し，余分な特徴を除去した抽出系列 E を求めて利用している．また，X を様々に選択すれば種々の抽出系列を求められる．

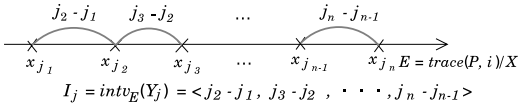

図 1 出現間隔列
Fig. 1 Interval sequence.

b. 特徴の例

まず，特徴 $Y_1, \cdots, Y_p$ の例を示した後で，特徴 X の選択例を述べる．定義 6 における特徴 $Y_1, \cdots, Y_p$ 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いことになる．たとえば，あるメソッドの呼び出しの存在がインライン化で除去されると，そのメソッド呼び出しの出現間隔列は変化する．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 メソッドや API クラスなどの API に関連する処理に相当する．なぜなら，API 関連の変換はそのライブラリの変換や解析が生じ，非常に困難だからである☆．よって，抽出系列 E がこれらの特徴を含むように，選択例として X を API メソッド呼び出しや API クラスのオブジェクト（以降，API オブジェクトと呼ぶ）を用いた処理（オブジェクト中のインスタンスフィールドを用いた動作やインスタンスメソッドの呼び出しの動作をここでは意味する．以降，これをオブジェクトの使用と呼ぶ）とし，各 $Y_1, \cdots, Y_p$ をある特定の API メソッド名または API クラス名を持つという特徴とする方法がある．

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，$E = trace(P,i)/X$，$E' = trace(P',i)/X$，$I_j = intv_E(Y_j)$，$I'_j = intv_{E'}(Y_j)$ $(1 \le j \le p)$ とし，出現間隔バースマーク $(I_1, \cdots, I_p)$ と $(I'_1, \cdots, I'_j)$ 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） $A = \langle a_1, \cdots, a_m \rangle$，$B = \langle b_1, \cdots, b_n \rangle$ の 2 系列の最長共通部分列の長さ[5]

☆ 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いる．

$L_{A,B}(m,n)$（以降，添字 A,B は省略）とは次による．

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

2 系列 A と B の一致率 $SR(A,B)$ を次で定める．

$$SR(A,B)=\frac{2\cdot L(m,n)}{m+n} \quad (1)$$

4.3 特徴のばらつきを利用した抽象化方法

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，処理の記述の違いから実装者によってこのばらつきは異なるであろう．これに着目し，このばらつきを表現する抽象化方法を提案する．

a. 抽象化方法

たとえば，$trace(P,i)/X=\langle a,x_1,b,c,d,x_2,x_3,e,f,x_4,g,x_5,\cdots\rangle$ を得たとする．ここで，$\{x_i\}_{i=1,2,\ldots}$ は $\{a,b,c,\cdots\}$ とは区別可能な特徴を持つ要素として，$k=4$ で考える．スライドしていく幅 $k+1$ の部分列の中に含まれる $\{x_i\}$ の個数の列（ここでは $\langle 1,2,2,2,2,3,2,2,\cdots\rangle$）を動的バースマークとするものである．次の定義で正確に述べる．

定義 8（*k-gram* 個数） 系列 $S=\langle s_1,\cdots,s_m\rangle$，$Y$ をある特徴，$S/Y=\langle s_{i_1},\cdots,s_{i_n}\rangle$ とおく．定数 k，r $(k\geq 0, 1\leq r\leq m-k)$ があって “$(i_{p-1}<r\leq i_p)\wedge(i_q\leq r+k<i_{q+1})$”（ただし，$i_0=0$，$i_{n+1}=m+1$ とおく）を満たす整数 p，q があるとき，S と区間 $[r,r+k]$ による Y の *k-gram* 個数を $num_S^{r,k}(Y)=q-p+1$ で定める．

定義 9（*k-gram* 個数列） 系列 $S=\langle s_1,\cdots,s_m\rangle$，ある特徴を Y，$K_r=num_S^{r,k}(Y)$ $(k\geq 0, 1\leq r\leq m-k)$ とおく．S と長さ k による Y の *k-gram* 個数列 $snum_S^k(Y)$ を次で定める．

$$snum_S^k(Y)=\langle K_1,\cdots,K_{m-k}\rangle$$

定義 10（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，$E=trace(P,i)/X$，$B_j=snum_E^k(Y_j)$ $(1\leq j\leq p)$ とおく．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．

$E=\langle x_1,\cdots,x_m\rangle$，$E/Y_j=\langle x_{j_1},\cdots,x_{j_t},\cdots,x_{j_n}\rangle$，$1\leq r\leq m-k$ とし，定義 10 における B_j の例を図示すると**図 2** のようになる．この図は r の位置から長さが k のバッファ内に含まれる特徴 Y_j を持つ要素が 3 個であるので，$num_E^{r,k}(Y_j)=3$ となる様子を示している．*k-gram* 個数列 B_j は r の値を 1 から $m-k$ まで変化させて，すなわち，図のようにバッファを系列上でシフトさせつつ逐次 *k-gram* 個数を列挙したものである．このように定義 10 は各特徴に対して *k-gram* 個数列を求めて，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いえる．個数を列挙する抽象化であるので，各特徴の出現順序の情報は捨象される．これにより，変換によって長さ k の範囲内で特徴の順序が多少変動しても，求められる *k-gram* 個数および *k-gram* 個数列は影響が少ないと考えられ，同時に動的バースマークも変換に保存性を持つと期待できる．

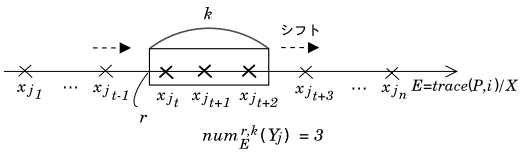

図 2 *k-gram* 個数
Fig. 2 *k-gram* number.

b. 特徴の例

定義 10 における各 $Y_1,\cdots,Y_p$ が変換されやすいと各 *k-gram* 個数列 B_j も変更されやすいことになる．そのため，4.2 節で述べた特徴の選択例と同様に，$Y_1,\cdots,Y_p$ は API 関連の特徴にし，X は API メソッド呼び出しや API オブジェクトの使用とするとよい．

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，k $(\geq\ 0)$ を定数とする．そして，$E=trace(P,i)/X$，$E'=trace(P',i)/X$，$B_j=snum_E^k(Y_j)$，$B'_j=snum_{E'}^k(Y_j)$ $(1\leq j\leq p)$ 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 つの比較には式 (1) を用い，一致率 $Q_j=SR(B_j,B'_j)$ 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．

4.4 特徴間の位置関係を利用した抽象化方法

プログラムでは一連の処理のまとまりで，ある仕事を行う．この一連の処理の中で互いに近い時点で実行される処理どうしは関連を持ち，依存している場合が多い．機械的な変換で各処理の実行時点の間隔を互いに大幅に引き離すことは，この依存関係を破壊して仕様を満たさなくする可能性があるため，一般に困難である．また，実装者によって処理の記述が異なれば，処理間のこの関係も異なるであろう．これらの点に着目し，実行時情報中に互いに近い位置に出現する特徴どうしの関係を集合で表現する抽象化を提案する．

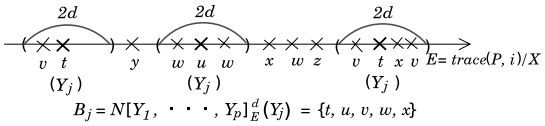

図 3 近傍集合
Fig. 3 Neighborhood set.

a. 抽象化方法

たとえば，$trace(P,i)/X = \langle a, x_1, b, c, d, x_2, x_3, e, f, x_2, g, x_1, \cdots \rangle$ を得たとする．ここで，各 x_i は区別可能な特徴を持つ要素として，$d = 2$ で考える．x_1 の出現箇所の距離 d 以下の周辺に現れる $\{x_i\}_{i=1,2,\ldots}$ の部分集合は $\{ x_1, x_2 \}$ であり，x_2 の周辺には $\{ x_1, x_2, x_3 \}$ があり，x_3 の周辺には $\{ x_2, x_3 \}$ がある．これらの部分集合の集まりを動的バースマークとしようとするものであるが，要素ではなく特徴ごとにまとめるものとして次の定義で正確に述べる．

定義 11（近傍集合） 系列 $S = \langle s_1, \cdots, s_m \rangle$，各 $Y, Y_1, \cdots, Y_p$ をある特徴，$S/Y = \langle s_{i_1}, \cdots, s_{i_n} \rangle$ とおく．$F(s_j)$ $(1 \leq j \leq m)$ は要素 s_j が $Y_1, \cdots, Y_p$ のいずれかの特徴を持つならば s_j，そうでなければ空を表すとする．d（≥ 0）を定数とし，S と範囲 d，および特徴 $Y_1, \cdots, Y_p$ による特徴 Y の近傍集合 $N[Y_1, \cdots, Y_p]^d_S(Y)$ を次で定める．

$$N[Y_1, \cdots, Y_p]^d_S(Y) = \{F(s_j) : 1 \leq j \leq m, 1 \leq \exists k \leq n, |j - i_k| \leq d\}$$

定義 12（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，$E = trace(P,i)/X$，$S_j = N[Y_1, \cdots, Y_p]^d_E(Y_j)$ $(1 \leq j \leq p)$ 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．

図 3 に定義 12 における S_j の例を示す．この図は系列 E 中に $Y_1, \cdots, Y_p$ の少なくとも 1 つの特徴を持つ要素 t，u，v，w，x，y，z がそれぞれあり，t と u は特徴 Y_j を持つとしている．要素 t と u の周囲を範囲 d 内で調べると，要素 t，u，v，w，x があるので，$B_j = \{t, u, v, w, x\}$ となる．このように定義 12 では集合としての抽象化によって，各特徴の出現順序の情報は捨象される．こうして，この出現順序が変換によって d の範囲内で多少変動しても，構成される動的バースマークには影響が少ないと期待できる．

b. 特徴の例

定義 12 における各 $Y_1, \cdots, Y_p$ が変換されやすい，たとえば実行時情報中から除去されてしまうと，各近傍集合 S_j の形態も変更されやすいことになる．そのため，4.2 節と 4.3 節で述べた特徴の選択例と同様に $Y_1, \cdots, Y_p$ は API 関連の特徴にする．X は API メソッド呼び出しや API オブジェクトの使用とするのがよいであろうが，ユーザ定義のものを含めることも可能である（評価実験ではユーザ定義のものも含めた）．

c. 動的バースマークの比較方法

近傍集合バースマークどうしの比較には以下の定義 13 を用いる．この定義における $NR(B, B')$ 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 13（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，各 $X, Y_1, \cdots, Y_p$ 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．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．

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

5. 評価実験

本章では，4 章で新たに提案した抽象化方法を用いて動的バースマークを実際に構成して行った評価実験について述べる．

5.1 動的バースマークの評価方法

動的バースマークの評価は保存性と弁別性の確認で一般に行われる．我々は保存性の確認に Java プログラムに種々の難読化を施すことが可能な SandMark [6] を用い，プログラムの難読化前とその後の動的バースマークの一致を確認することにした☆．この場合，一致があるほど保存性を満たすといえる．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複数用意し，それらから構成される動的バースマークどうしの一致を調べた．この場合，一致しないほど弁別性を満たすといえる．

5.2 保存性の確認結果

jdepend-2.9.1.jar，java2html.jar，tar.jar の 3 個のツールプログラム☆☆のそれぞれを対象とし，構成され

☆ SandMark はソフトウェアプロテクションの研究用に開発され公開されているツールで，難読化についてはレイアウト，制御，データなどに関する異なる種類の種々の手法が実装されている．これを用いることで種々の難読化に対する保存性を調べることができると考えた．

☆☆ それぞれ，http://www.clarkware.com/software/JDepend.html，http://www.java2html.de/，http://www.trustice.com/java/tar/．

表 1 出現間隔バースマークの保存性
Table 1 Resilience of interval birthmarks.

難読化方法	jdepend-2.9.1.jar		java2html.jar		tar.jar	
	メソッド	フィールド	メソッド	フィールド	メソッド	フィールド
Array Splitter	0.33	0.43	0.30	0.00	0.17	0.40
Class Splitter	1.00	0.00	1.00	0.14	1.00	0.20
Dynamic Inliner	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	0.00	1.00	0.10	1.00	0.00	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
全 33〜36 種の平均	0.911	0.956	0.912	0.944	0.894	0.945

る出現間隔バースマーク，スライドバッファバースマーク，近傍集合バースマークの難読化の保存性を調べた．以降，対象としたプログラムを 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保存性

定義 6 における特徴 X を API メソッド呼び出し，$Y_1, \cdots, Y_p$ をある 15 個の API メソッド名を持つ特徴（$p = 15$）とし，P と P' から構成される出現間隔バースマークどうしを比較した．その比較結果は，$E = trace(P, i)/X$，$E' = trace(P', i)/X$，$I_j = intv_E(Y_j)$，$I'_j = intv_{E'}(Y_j)$ 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）で **表 1** のメソッドの列に示す．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，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，$Y_1, \cdots, Y_p$ をある 7 個の API オブジェクトの使用（$p = 7$）として出現間隔バースマークを構成し，同様に比較した結果を表 1 のフィールドの列に示す．なお，難読化に失敗し評価できなかったものがいくつかあり，表では×印で示す．この表から，対象プログラムが変わってもほぼ同じ傾向の数値を示していることが分かる．他の動的バースマークの実験でも同じ傾向を示したので，以降の表では jdepend-2.9.1.jar に対する結果を主に示す．

☆ 盗用の関係にある 2 つのプログラム間で出現間隔列および *k-gram* 個数列を比較するとほとんどの場合，一致率は 0.8 から 1.0 の数値を示す．そのため，ここでの数値を 0.8 とした．

☆☆ SandMark の持つ難読化のうち暗号利用などの 2 つを除く 37 種類の難読化について実験を行っているが，紙数の関係で代表的と考えられる 11 種について数値を示し，全体の単純平均を最終行に示している．各難読化の詳細は文献 3)，6) を参照されたい．

表 2 スライドバッファバースマークの保存性
Table 2 Resilience of slide buffer birthmarks.

難読化方法	jdepend-2.9.1.jar	
	メソッド	フィールド
Array Splitter	0.40	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	0.27	1.00
Publicize Fields	1.00	1.00
Split Classes	1.00	1.00
Static Method Bodies	1.00	1.00
全 36 種の平均	0.928	0.956
java2html.jar 平均（33 種）	0.915	0.952
tar.jar 平均（33 種）	0.899	0.952

表からいくつかの難読化に保存性が低い．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て新たに追加する．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．すなわち，これらはメソッド呼び出しやフィールド変数値の参照を余分に追加するため保存性が低くなると考えられる．一方で，他の難読化には保存性があるといえる．

5.2.2 スライドバッファバースマークの保存性

定義 10 における各特徴 $X, Y_1, \cdots, Y_p$ を 5.2.1 項と同じにし，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て比較した．この比較も 5.2.1 項と同様に，p 個中 n 個が 0.8 以上の一致率の場合，n/p の値で結果を**表 2** に示す．なお，定義 10 における k の値を 2，4，6，8，10 として実験したが，大きな相

表 3 近傍集合バースマークの保存性
Table 3 Resilience of neighborhood set birthmarks.

	jdepend-2.9.1.jar	
難読化方法	メソッド	フィールド
Array Splitter	0.90	0.90
Class Splitter	1.00	0.85
Dynamic Inliner	0.85	1.00
Inliner	0.99	1.00
Method Merger	1.00	1.00
Objectify	1.00	1.00
Overload Names	1.00	1.00
Promote Primitive Types	0.89	1.00
Publicize Fields	1.00	1.00
Split Classes	1.00	0.00
Static Method Bodies	0.78	1.00
全 36 種の平均	0.977	0.965
java2html.jar 平均（33 種）	0.973	0.987
tar.jar 平均（33 種）	0.978	0.971

表 4 出現間隔バースマークの弁別性
Table 4 Credibility of interval birthmarks.

課題 1			課題 2					
A,*B*	*B*,*C*	*C*,*A*	*A*,*B*	*A*,*C*	*A*,*D*	*B*,*C*	*B*,*D*	*C*,*D*
0.00	0.00	0.22	0.00	0.17	0.00	0.00	0.00	0.00

表 5 スライドバッファバースマークの弁別性
Table 5 Credibility of slide buffer birthmarks.

課題 1			課題 2					
A,*B*	*B*,*C*	*C*,*A*	*A*,*B*	*A*,*C*	*A*,*D*	*B*,*C*	*B*,*D*	*C*,*D*
0.00	0.00	0.22	0.00	0.17	0.10	0.00	0.00	0.00

表 6 近傍集合バースマークの弁別性
Table 6 Credibility of neighborhood set birthmarks.

課題 1			課題 2					
A,*B*	*B*,*C*	*C*,*A*	*A*,*B*	*A*,*C*	*A*,*D*	*B*,*C*	*B*,*D*	*C*,*D*
0.03	0.03	0.18	0.07	0.34	0.24	0.09	0.05	0.19

違は見られなかったので $k = 4$ のときの結果を示している．5.2.1 項と同様の理由で保存性の低い難読化があるが，ほとんどには保存性があるといえる．

5.2.3 近傍集合バースマークの保存性

定義 12 における特徴 X をメソッド呼び出し（ユーザ定義メソッドと API メソッドの両方のメソッド呼び出し），$Y_1, \cdots, Y_p$ を抽出系列 $E = trace(P, i)/X$ 中に含まれるすべての API メソッド名を持つ特徴とし，P と P' から 2 つの近傍集合バースマークを構成して比較した．比較結果は式 (2) の一致率（小数点第 3 位を四捨五入）で**表 3** のメソッドの列に示す．なお，定義 12 における d の値を 2，4，6，8，10 として実験したが，大きな相違は見られなかったので $d = 4$ のときの結果を示している．また，X をフィールド変数値の参照（フィールド変数は API オブジェクトを保持するか否かにかかわらない），$Y_1, \cdots, Y_p$ を E 中に含まれるすべての API オブジェクトの使用とし，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 3 のフィールドの列に示す．

表から Split Classes の難読化に保存性が低いことが分かる．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，表のフィールド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保存性があるといえる．

5.3 弁別性の確認結果

異なる 3 人（*A*，*B*，*C*）に課題 1☆のプログラムを，異なる 4 人（*A*，*B*，*C*，*D*）に課題 2☆☆のプログラムをそれぞれ実装させ，これらから構成される出現間隔バースマーク，スライドバッファバースマーク，近傍集合バースマークどうしの一致を調べた．作られたプログラムは 200 行未満の小規模なものである．5.2.1〜5.2.3 項と同様に，特徴 $X, Y_1, \cdots, Y_p$ としてメソッド呼び出しおよびいくつかのメソッド名を選択し，一致率も同様の方法で算出した．これらの実験結果を**表 4**，**表 5**，**表 6** に示す（*A* と *B* の動的バースマークの比較結果は表の *A*，*B* の列に示す）．

課題 1 に対するプログラムでは実装に用いたクラスがそれぞれ異なっており，プログラム間で共通のメソッドの数が少ないため一致率が低く示されたと考えることもできる．実装者が異なれば利用クラスも異なることもバースマークを構成する情報として有用であるが，利用クラスが偶然に一致する場合も考えられる．この状況を考慮して課題 2 では，実装に必要と思われるハッシュや動的配列を利用する場合には指定されたクラス☆☆☆を利用するという制限をつけて実験を行った．5.2 節で示した各表の数値と比べると著しく低い値が得られた結果となり，これらの動的バースマークは弁別性を十分に満たしているといえる．

6. 実験結果の考察

5 章での実験の結果から枠組みの有効性について考察する．実験を行った動的バースマークは 5.2 節で示した各表から，多くの難読化手法を実装している SandMark の持つほとんどの難読化に保存性を持つといえる．多くの難読化手法に対する実験ではあるが必

☆ 英文と辞書を入力し，辞書に存在しない単語を出力する．
☆☆ 英文を入力し，出現回数が上位 3 つの単語についてその直後に出現する単語を出現回数順（同じであれば辞書式順序）で出現回数とともにすべて出力する．
☆☆☆ それぞれ，java.util.Hashtable，java.util.ArrayList．

ずしも十分であるとはいえず，結論を出すには限界があるが，多くの場合に変換に保存性のある動的バースマークを構成できることは強く予想できる．動的バースマークと難読化のいくつかの組については保存性を示していないが，そのような難読化に対しても別の動的バースマークでの結果を見ると保存性は高い．たとえば，表 1 と表 2 では Array Splitter の難読化に対し低い数値であるが，表 3 では高い数値である．特定の動的バースマークと難読化の組に対して保存性をある程度予測できる場合もあるが，一般には困難であろうし未知の変換に対しては不可能である．しかし，ここでの実験結果から互いの弱点を補う関係にある動的バースマークを系統的に構成できる可能性が高いことが示されている．

さらに，近傍集合バースマークの実験においては，抽出特徴を従来では変換に保存性が低いとされたメソッド呼び出しとフィールド変数値の参照とした（API と関連しないユーザ定義のメソッドとフィールド変数は一般に保存性が低い）．それにもかかわらず，表 3 からほぼすべての難読化に対して高い保存性を示している．これは抽象化により，変換されやすい箇所が効果的に捨象された結果といえる．単一の動的バースマークでは保存性が低くても，複数の動的バースマークを併用することでそれぞれの弱点を補完できることが予想できる．

また，実験した各動的バースマークが弁別性を満たすことは，利用するクラスが偶然一致するという状況を考慮して設定された課題に対する実験も含め，確認できた．

こうして，ここで提案した動的バースマークを構成する枠組みに従うことで，すなわち実行時情報から抽出する特徴と抽象化とを別個に考え組み合わせることで，定義 2 を満たす複数の動的バースマークを容易に構成できることが示されたといえる．

7. おわりに

本稿では実行時情報からの特徴の抽出とその抽象化の結果を動的バースマークと考える枠組みを扱った．この枠組みでは，抽出する特徴の種類と抽象化の種類の独立な組合せにより種々の有効な動的バースマークの構成が期待できる．我々は抽象化の例として，繰返しパターンによる動的バースマーク，出現間隔バースマーク，スライドバッファバースマーク，近傍集合バースマークを提案し，メソッド呼び出しとフィールド変数値の参照の特徴を抽出して枠組みに従った動的バースマークをいくつか構成する実験を行った．その結果，実際に種々の有効な動的バースマークを構成できた．

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抽出特徴を本稿で示したものだけではなく，その他の種々の特徴を抽出することの検討が必要である．抽象化方法も同様に本稿で示した以外にもそのほかの方法を検討し，より多数の抽出特徴と抽象化方法の組合せが実現可能になるとよいであろう．バースマーク構成の枠組みの評価をさらに進める必要はあり，構成方法の検討とともにそれらの依存関係を明らかにすることは重要であると考える．また，動的バースマークの手法はプログラム全体の盗用だけでなく，パッケージ単位やクラス単位での盗用に対しても対応できることが望まれる．盗用状況が種々ありうるために一般には困難と考えられるが，盗用の程度と疑わしい盗用範囲の識別可能性の組合せを設定し順次検討を進めていきたいと考えている．

参考文献

1) Myles, G. and Collberg, C.S.: Detecting Software Theft via Whole Program Path Birthmarks, *Proc. 7th International Security Conference* (*ISC 2004*), LNCS 3225, pp.404–415 (2004).
2) 岡本圭司，玉田春昭，中村匡秀，門田暁人，松本健一：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，情報処理学会第 46 回プログラミング・シンポジウム報告集，pp.41–50 (2005).
3) Collberg, C.S. and Thomborson, C.: Watermarking, Tamper-Proofing, and Obfuscation – Tools for Software Protection, *IEEE Trans. Softw. Eng.*, Vol.28, No.8, pp.735–746 (2002).
4) 古田壮宏，真野芳久：実行系列の抽象表現を利用した動的バースマーク，電子情報通信学会論文誌 D-I，Vol.J88-D1, No.10, pp.1595–1598 (2005).
5) Hirschberg, D.S.: A Linear Space Algorithm for Computing Maximal Common Subsequences, *Comm. ACM*, Vol.18, No.6, pp.341–343 (1975).
6) SandMark: A Tool for the Study of Software Protection. http://sandmark.cs.arizona.edu/

(平成 18 年 3 月 17 日受付)
(平成 19 年 1 月 9 日採録)

林　晃一郎（学生会員）

2006 年南山大学大学院数理情報研究科数理情報専攻博士前期課程修了．同年 4 月株式会社インテック入社．ソフトウェアプロテクションの研究に従事．

楓　　基靖

南山大学大学院数理情報研究科数理情報専攻博士前期課程に在学中．ソフトウェアプロテクション，Java プログラムのためのトレーサやデバッガの研究に従事．

真野　芳久（正会員）

1971 年京都大学理学部卒業．同年電子技術総合研究所入所．1991 年南山大学経営学部教授，2000 年南山大学数理情報学部教授．工学博士．プログラミング支援，データ圧縮，ソフトウェアプロテクションの研究に従事．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会各会員．