

Notation for Watermarking

Stephen Drape and Clark Thomborson

Draft of 26th June 2006

1 Basics

We write \mathbb{B} to denote the Boolean type which contains two members denoted by T and F and \mathbb{N} represents the set of natural numbers. We use $:$ for type declarations and so $x : T$ means “ x has type T ”.

We use the following pairs of brackets: $\{\dots\}$ for sets, (\dots) for tuples, $\langle \dots \rangle$ for sequences and $[\dots]$ for lists. In general, as sets are unordered, we cannot “access” a specific element of a set but we can access values in tuples and sequences. If t is the n -tuple

$$t = (t_1, t_2, \dots, t_n)$$

then we write $t.j$ to access the j -th element and in this case $t.j = t_j$. Note that in Z , sequence access is usually written as functional application and so we would write $s\ j$ to access the j -th element of s . Instead we overload the dot notation and so if

$$s = \langle s_1, s_2, \dots, s_m \rangle$$

then $s.j = s_j$.

We write the type of function f with domain D and range R as

$$f : D \rightarrow R$$

We use \times in a type declaration to denote tupling. So $T : A \times B$ means that if $t \in T$ then $(\exists a \in A; b \in B) \bullet t = (a, b)$. For a set S we also will use the shorthand S^n to mean

$$\underbrace{S \times \dots \times S}_{n \text{ times}}$$

The expression $\text{Seq } S$ denotes the set of sequences over S and similarly $\text{List } S$ the set of lists over S . The expression $\mathbb{P}S$ represents the power set of a set S .

A string is defined to be a list of characters and we take “abc” to mean $['a', 'b', 'c']$. We define

String : List Char

We denote $[a..b]$ (where $0 \leq a < b$) to be the following set of natural numbers:

$$\{n \mid n \in \mathbb{N} \wedge a \leq n \wedge n \leq b\}$$

The symbol $=$ means equality and \triangleq the definition of a function.

2 Sets

We denote the set of all (Java) programs by \mathcal{P} and \mathcal{P}_E to be a finite, ordered subset of \mathcal{P} which represents a set of programs that have been selected for experimentation. Since \mathcal{P}_E is ordered we can choose the j -th element of \mathcal{P}_E (where $j \in [1..|\mathcal{P}_E|]$) and we denote the j -th element by ψ_j .

A *watermark* is taken to be a string of characters. Let \mathcal{W} denote the set of watermarks and so $\mathcal{W} \subseteq \mathbb{P}\text{String}$. If we allow any string to be a watermark then the inclusion becomes equality otherwise we need to specify which strings are allowable as watermarks.

We write \mathcal{W}_E to be a finite, ordered subset of \mathcal{W} which has been selected for experimentation. Since \mathcal{W}_E is ordered we can specify watermarks $\omega_1, \omega_2, \dots, \omega_n$ so that

$$\mathcal{W}_E = \{\omega_1, \omega_2, \dots, \omega_n\}$$

We use a special element $\omega^0 \notin \mathcal{W}$, called the *null watermark*, to model cases where a program has not been watermarked (this will be used in the definition of an accuracy function in Section 3.3). We can augment \mathcal{W} to include this special watermark and so we define

$$\mathcal{W}^0 \triangleq \mathcal{W} \cup \{\omega^0\}$$

The set of all keys which can be used to embed an watermark is denoted by \mathcal{K} . The set \mathcal{K}_E is a finite, ordered subset of \mathcal{K} and contains keys which have been selected for experimentation. We write κ_j to denote the j -th element of \mathcal{K}_E (which we can do as \mathcal{K}_E is ordered).

2.1 Experiments

An experimental unit consists of a program (drawn from our experimental set \mathcal{P}_E), a watermark (from \mathcal{W}_E^0) and a key (from \mathcal{K}_E). Let \mathcal{U} denote the set of experimental units then

$$\mathcal{U} : \mathbb{P}(\mathcal{P} \times \mathcal{W}^0 \times \mathcal{K})$$

and so if $\mu \in \mathcal{U}$ then

$$\mu : \mathcal{P} \times \mathcal{W}^0 \times \mathcal{K}$$

We may want to define \mathcal{U} to contain a specific set of experimental units and so we will have to restrict \mathcal{U} so that it is just a finite, ordered subset of $\mathbb{P}(\mathcal{P} \times \mathcal{W}^0 \times \mathcal{K})$. The size of \mathcal{U} has an upper limit

$$|\mathcal{U}| \leq |\mathcal{P}_E| \times |\mathcal{W}_E^0| \times |\mathcal{K}_E|$$

To find (say) the first component of an experiment μ then we write $\mu.1$. As a shorthand, we will write (i_1, i_2, i_3) to denote the unit $(\psi_{i_1}, \omega_{i_2}, \kappa_{i_3})$. We can define a choice function

$$ch : \mathbb{N}^3 \rightarrow \mathcal{U}$$

which maps tuples of integers onto units as follows:

$$ch(i_1, i_2, i_3) \triangleq (\psi_{i_1}, \omega_{i_2}, \kappa_{i_3})$$

An experiment is a sequence of experimental units and if \mathcal{X} is the set of experiments then

$$\mathcal{X} : \text{Seq } \mathcal{U}$$

Thus if $\chi \in \mathcal{X}$ then

$$\chi = \langle \mu_1, \mu_2, \dots, \mu_m \rangle$$

where $(\forall j \in [1..m]) \bullet \mu_j \in \mathcal{U}$. To find the j -th element of χ then we write $\chi.j$ and so (say) the watermark in the j -th element of χ can be written as $(\chi.j).2$.

3 Functions

In this section we define various functions which will help to model the process of embedding and detecting watermarks.

3.1 Embedding function

An embedding function em takes a program, a watermark and a key to produce a new program — the *watermarked program*. The function em has the type:

$$em : \mathcal{P} \times \mathcal{W}^0 \times \mathcal{K} \rightarrow \mathcal{P}$$

We may define \mathcal{E} to be the set of embedding functions. An embedding function $em \in \mathcal{E}$ should have the following two properties. Firstly, a program is invariant under the null watermark:

$$(\forall \psi \in \mathcal{P}; \kappa \in \mathcal{K}) \bullet em(\psi, \omega^0, \kappa) = \psi$$

Secondly, the embedding process outputs a program which is functionally equivalent to the original program:

$$(\forall \psi \in \mathcal{P}; \omega \in \mathcal{W}; \kappa \in \mathcal{K}) \bullet \psi \equiv em(\psi, \omega, \kappa)$$

where \equiv means functional equivalence for some semantics (which are not defined here).

3.2 Extraction and detection functions

The extraction function ex takes a watermarked program and a key and returns a watermark. The type of this function is:

$$ex : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{W}^0$$

A program ψ can be successfully watermarked with ω under key κ if

$$ex(em(\psi, \omega, \kappa), \kappa) = \omega$$

A detection function d is intended to detect the presence of a watermark — it returns T if it finds a watermark and F otherwise. The function d has the following type:

$$d : \mathcal{P} \rightarrow \mathbb{B}$$

The symbol \mathcal{D} denotes the set of all detectors.

3.3 Accuracy function

An accuracy function is used to measure how accurate a detection function is at detecting the presence of a watermark. For a detector $d \in \mathcal{D}$, an accuracy function ρ has the following type:

$$\rho : \mathcal{D} \times \mathcal{E} \times \mathcal{P} \times \mathcal{W} \times \mathcal{K} \rightarrow (\mathbb{B} \times \mathbb{B})$$

We can define ρ as follows:

$$\rho(d, e, \psi, \omega, \kappa) \triangleq (\omega \neq \omega^0, d(e(\psi, \omega, \kappa))) \quad (1)$$

From ρ we have four possible outcomes:

- (T, T) : True Positive (TP) — the detector correctly identifies that the program has a watermark
- (T, F) : False Negative (FN) — the detector fails to identify that the program has a watermark
- (F, T) : False Positive (FP) — the detector incorrectly identifies that the program has a watermark
- (F, F) : True Negative (TN) — the detector correctly identifies that the program does not have a watermark

3.4 Rates

For a particular experiment we may want to count (say) the number of True Positives obtained for a particular detector and extractor during the run of an experiment. We can define a function

$$TP : \mathcal{X} \times \mathcal{D} \times \mathcal{E} \rightarrow \mathbb{N}$$

which counts the number of true positives obtained over the run of an experiment with respect a particular detector and embedding function. We can define TP as follows:

$$TP(\chi, d, em) \triangleq |\{i \in [1..|\chi|] \mid \rho(d, e, (\chi.i).1, (\chi.i).2, (\chi.i).3) = (T, T)\}| \quad (2)$$

So the true positive rate for the experiment χ can be defined as follows:

$$rate_{TP} \triangleq \frac{TP(\chi, d, em)}{|\chi|} \quad (3)$$

Rates for the other outputs of ρ (*i.e.* TN, FP and FN) can be defined similarly.

3.5 Grams

The set of possible Java opcodes is denoted by \mathcal{A} and this is used as an alphabet in k -gram analysis. The notation \mathcal{A}^k denotes the strings of length k over the alphabet \mathcal{A} (the so-called k -grams). The function $g_{\#}$ counts the number of occurrences of a particular k -gram in a program:

$$g_{\#} : \mathcal{A}^k \times \mathcal{P} \rightarrow \mathbb{N}$$

The function $grams$ should return the set of k -grams for a particular program

$$grams : \mathcal{P} \times \mathbb{N} \rightarrow \mathbb{P} \mathcal{A}^k$$

where

$$grams(\psi, k) \triangleq \{x \in \mathcal{A}^k \mid g_{\#}(x, \psi) > 0\}$$

Since $grams$ returns a set then duplicates are discarded.

So if $p \equiv \langle \text{IADD}, \text{IADD}, \text{ISUB}, \text{IADD} \rangle$ then

$$\begin{aligned} grams(\psi, 2) &= \{\langle \text{IADD}, \text{IADD} \rangle, \langle \text{IADD}, \text{ISUB} \rangle, \langle \text{ISUB}, \text{IADD} \rangle\} \\ grams(\psi, 1) &= \{\langle \text{IADD} \rangle, \langle \text{ISUB} \rangle\} \end{aligned}$$

Note that every k -gram occurring in a program has a count of at least 1:

$$(\forall k \in \mathbb{N}; \psi \in \mathcal{P}; x \in \mathcal{A}^k) \bullet x \in grams(\psi, k) \Rightarrow g_{\#}(x, \psi) \geq 1$$

3.6 An enhanced detector

We can enhance a detector so that it will detect whether a program has a specific watermark. An enhanced detector d^* has type

$$d^* : \mathcal{P} \times \mathcal{W}^0 \times \mathcal{K} \rightarrow \mathbb{B}$$

and can be defined, for some extractor ex , as

$$d^*(\psi, \omega, \kappa) \triangleq ex(\psi, \kappa) = \omega$$

where $\psi \in \mathcal{P}, \omega \in \mathcal{W}^0$ and $\kappa \in \mathcal{K}$.

If we use this enhanced detector then we also need to change the definitions of ρ and the rate functions.

4 Modelling failures

The detector function may not terminate normally so we can add an extra case \perp (called *bottom*) to the range that models cases such as abnormal termination or non-termination. We write \mathbb{B}_{\perp} to denote the type which contains three elements: T , F and \perp . Thus such a detector function has the type:

$$d_{\perp} : \mathcal{P} \rightarrow \mathbb{B}_{\perp}$$

We write \mathcal{D}_{\perp} to denote the set of all detector functions which can return \perp . We can define a type for enhanced detector functions (from Section 3.6) similarly.

Since a detector function d_{\perp} can return \perp we will need to change the type of an accuracy function that uses a detector from \mathcal{D}_{\perp} :

$$\rho_{\perp} : \mathcal{D}_{\perp} \times \mathcal{E} \times \mathcal{P} \times \mathcal{W} \times \mathcal{K} \rightarrow (\mathbb{B} \times \mathbb{B}_{\perp})$$

The definition of ρ_{\perp} is the same as in Equation (1).

We also need to change the definition of *rate* — Equation (3) — so that it takes account of the fact that the detector can produce an output of \perp . For example, we can define

$$rate_{TP_{\perp}} \triangleq \frac{TP_{\perp}(\chi, d_{\perp}, em)}{|\{i \in [1..|\chi|] \mid d_{\perp}(e((\chi.i).1, (\chi.i).2, (\chi.i).3)) \neq \perp\}|}$$

The function TP_{\perp} is similar to Equation (2) with the appropriate change of type. The denominator in definition of $rate_{TP_{\perp}}$ counts the number of experiments of χ in which the detector successfully terminates.