Formal Aspects of Anonymity

Robert Morelli

UUCS-02-006

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

January 15, 2002

Abstract

We present a formal definition of anonymity in the context of concurrent processes. The definition is given in category theoretic terms. Moreover, the concept of a split cofibration is shown to both simplify the analysis of anonymity as well as to increase the framework's expressiveness. Because of its categorical nature, our definition is largely independent of any specific model of concurrency. We instantiate the theory to two specific models, Hoare trace languages and probabilistic transition systems. By providing a semantics for CSP, the former model endows CSP with a definition of anonymity that applies to every CSP process simulation. The latter model, probabilistic transition systems, provides a definition of anonymity applicable to probabilistic processes. This seems to be the first general such definition.

I. Introduction

Anonymity is a desirable feature for many kinds of transactions, including voting, payments, communications, and publishing. The existing research on anonymity is roughly clustered along two axes. On the one hand, there is an extensive literature on security, including formal analyses of aspects of security such as integrity and secrecy. On the other, there is much work specific to anonymity, including informal descriptions of useful anonymous systems and practical techniques for implementing them. However, the literature is relatively sparse at the intersection, leaving a gap where one would hope to have a richer body of theory specific to anonymity. In this report we present a contribution to filling this gap by giving a formal definition of anonymity applicable to a broad class of systems.

Most existing anonymous systems provide services over computer networks. In the prototypical example, a system allows Alice to transmit a message to Bob across a network without it being apparent that she has done so, even to an attacker who monitors all network traffic. However, it is useful to view the concept of anonymity in a broader context. For instance, a local voting system, without any networked communication, may still require a form of anonymity, namely unlinkability between voter and vote. Indeed, even in a network communications system, there are a number of distinct senses of anonymity; Pfitzmann and Waidner [PW87] distinguish sender anonymity, receiver anonymity, and unlinkability between sender and receiver. On a deeper level, any concept of anonymity confined to a networking model will inevitably run up against artificial limitations.

In this report, we consider anonymity within the general context of concurrency. Accordingly, we view an anonymous system as a process, a system built out of concurrently interacting entities. In casting our definition in terms of processes, we are able to reap the rewards of several decades of research on concurrency. For example, process algebras such as CSP and the π-calculus are Turing complete, so they are able to model any computationally meaningful system, and are amenable to formal analysis. At the same time, these algebras are well suited to intuitive, high-level specifications of real systems and directly model fundamental concepts such as nondeterminism and concurrency. Moreover, extensions of these traditional process algebras can express behavior such as probabilistic choice, priority, time, traffic patterns, etc., opening the way to increasingly expressive definitions of anonymity. For instance, we show in section V how to apply our definition to a probabilistic model of concurrency. In fact, ours seems to be the first definition of anonymity capable of handling probability.

A large number of models of concurrency have been proposed. Our approach is largely independent of any specific model and seems to apply to many of the models in common use. This generality is achieved by formulating the definition in category theoretic terms. At this level of abstraction, the idea of anonymity becomes very simple; it is simply the set of symmetries of a simulation between processes: $k : \mathit{System} \to \mathit{Screen}$. The two related processes are the anonymous system itself, *System*, and an attacker's knowledge of it, *Screen*. This latter knowledge is modeled as a second system that evolves in parallel with the original system (but which usually has a coarser set of states). Intuitively, the simulation k drives *Screen*.

In conformity with Kerckhoffs' principle, the attacker is assumed to have full knowledge of the specification and implementation of *System*, *Screen*, and k, but is only granted direct knowledge of the state of *Screen*, not that of *System*. Because all deductions the attacker makes about the state of *System* are dependent on k, constraints on k place constraints on the attacker's knowledge, regardless of the attacker's inference system. In particular, the requirement of symmetry guarantees that it is only possible for an attacker to make general deductions about all individuals equated by symmetries. It is impossible to make any deduction that singles out one such individual from others. This is the essence of anonymity.

The set of symmetries of the knowledge simulation k is called the *anonymity monoid.* (In contrast to most contexts in which symmetry is used, we do not require our symmetries to be invertible because there are some contexts in which noninvertible symmetries are meaningful.) Depending on the desired anonymity requirements of a system, the structure demanded may vary. Roughly speaking, the larger the anonymity monoid, the greater the guaranteed anonymity of the system, but the structure of the anonymity monoid carries finer information. For instance, a voting system might demand symmetry between all voters. In other words the anonymity monoid is required to contain the full symmetric group on the set of voting events. However, suppose that the voters are divided into parties P_1 and P_2 and we demand symmetry between members within a party but not between parties. In other words, an attacker may be able to distinguish votes originating in P_1 from those originating in P_2, but cannot distinguish individual votes within those groups. This requirement demands a certain subset of the full symmetric group. Suppose, for another example, that a voting system only exhibits symmetry with respect to a cyclic group of permutations around a cyclic ordering. This provides a much weaker notion of anonymity, since it allows for an attacker to deduce the distance between the positions of two voters in the cyclic ordering. Evidently, these are only two of many possible variations, corresponding to the many kinds of anonymity which a voting system may require.

We are aware of two prior contributions[1] to the problem of formalizing anonymity, [SS96] and [SS99]. Of these, the paper by Schneider and Sidiropoulos [SS96] is more directly relevant to our approach and was in fact one of our inspirations. However, that work had a number of limitations. First, being set within the language of Hoare's CSP process algebra [Hoa85], it was bound to a single process algebra and inherited that algebra's expressive limitations. In particular, CSP cannot express probabilistic behavior, a serious limitation in modeling anonymity. For instance, the analysis in [SS96] will count as anonymous a system in which, say, the publisher of a document can always be determined with a probability of 99%, but never with absolute certainty. Another limitation is that their system cannot handle the anonymity of dependent events, such as the voting events in a system in which each voter can vote at most once. Another formal approach to anonymity, based on logic, was proposed in [SS99]. It is largely complementary to our own approach. However, like [SS96], it is *possibilistic* rather than probabilistic.

II. A General Framework for Anonymity

Our formalization of anonymity is set in the general context of concurrent processes. We will first describe its ingredients in broad terms. As was indicated in the introduction, anonymity is a function of both a system and an attacker's knowledge of that system. The system is modeled as a process *System* and the attacker's knowledge is modeled as a simulation $k : \mathit{System} \to \mathit{Screen}$. For this we require a model $\mathcal{P}$ of concurrent processes which includes a notion of simulation. Many such models have been studied: Petri nets, labelled transition systems, pomsets, etc. We also require a notion of the *elements* of a process. Roughly speaking, the elements of a process are the individuals or, more precisely, the specific actions, whose anonymity is under consideration. In order to bring elements into our framework we need both a model $\mathcal{M}$ for them as well as a function $\alpha : \mathcal{P} \to \mathcal{M}$ which associates elements $\alpha(P)$ to each process P. For example, if we base our model $\mathcal{P}$ of processes on CCS [Mil89], then we could take the elements of a process to be its alphabet of actions. Technically, the study of anonymity is greatly simplified when α is a *split cofibration*. Very roughly, this means that any simulation of processes $S \to T$ induces a mapping $\alpha(S) \to \alpha(T)$ on underlying elements, and conversely transforming the underlying elements of a process induces a new process simulating the original process. For instance, transforming the alphabet of a CCS process induces a new process by renaming.

Because of the proliferation of models of concurrency, it is advantageous to work at a higher level of abstraction. Following a line of research expounded in [WN95], we will view a model of concurrency as a category. In this framework, the objects of the category correspond to processes, the morphisms to simulations, and the ba-

[1] A third paper has recently come to our attention: Vitaly Shmatikov, Dominic J.D. Hughes, Defining Anonymity and Privacy. See http://www.dsi.unive.it/IFIPWG1_7/WITS2002/prog/annotated_program.html

sic constructions of concurrency theory appear as basic category theoretic structures appropriately interpreted. Further, the relationships between different models of concurrency typically appear as adjunctions.

Let us introduce some basic category theoretic notation and terminology. Let $\mathcal{C}$ be a category, which for convenience we will assume is locally small. If A and B are objects of $\mathcal{C}$, denote by $\mathrm{Hom}(A, B)$ the hom set of morphisms from A to B. If $f \in \mathrm{Hom}(A, B)$, then we say that A is the *source* of f and B is the *target* of f. For any object A, the hom set $\mathrm{Hom}(A, A)$ is a monoid which we denote $\mathrm{End}(A)$.

Let $\mathcal{P}$ be a category corresponding to some notion of process. We will call $\mathcal{P}$ the *process category*. Let $k : S \to N$ be a morphism in $\mathcal{P}$. We will call S the *system*, N the *screen*, and k the *knowledge morphism*.

Definition II.1 *A symmetry of $k : S \to N$ is an element $r \in \mathrm{End}(S)$ satisfying $k \circ r = k$.*

A symmetry of k is a transformation of the system S that leaves k unchanged. Intuitively, it is a transformation that k cannot detect, hence it is an *anonymous transformation*. Because we haven't yet introduced the elements of a process into the framework, this definition is more abstract than we'll typically want to apply in practice. We therefore defer further discussion until after fleshing out the framework with a fibering over a category of elements.

A. Anonymity in Cofibrations

In the category theoretic interpretation of concurrency, *fibrations* and *cofibrations* play an interesting role related to what we called above the *elements* of a process. Depending on the specific model, the elements of a process represent variously events, actions, names, ports, communication channels, locations, etc. On a simple level, extracting the elements of a process just amounts to a forgetful functor $\alpha : \mathcal{P} \to \mathcal{B}$, where $\mathcal{B}$ is typically a category of relatively unstructured objects, such as sets. However, in many interesting models, *cocartesian liftings* of morphisms in $\mathcal{B}$ correspond to relabelling operations for processes, while *cartesian liftings* correspond to restriction operations. In many models of concurrency, such liftings generally exist, supplying the functor α with fibration and cofibration structures. Only relabelling operations will play a role in this paper, so we will only need cocartesian liftings and cofibrations. What we need of these concepts is very briefly recalled in appendix A.

Let $\mathcal{P}$ be a category corresponding to some notion of process, with a split cofibration $\alpha : \mathcal{P} \to \mathcal{B}$ with splitting κ. We call $\mathcal{B}$ the category of *elements.*

Before stating the main definition, we need a bit of notation. Let S be an object of $\mathcal{P}$ as above and let $E = \alpha(S)$. Denote by $\mathrm{End}_\alpha(E)$ the submonoid of $\mathrm{End}(E)$ consisting of all $h \in \mathrm{End}(E)$ for which $h_*(S) = S$. Denote by $\mathrm{End}_\alpha(S)$ the submonoid of $\mathrm{End}(S)$ consisting of all $r \in \mathrm{End}(S)$ for which $\alpha(r) \in \mathrm{End}_\alpha(E)$. Note that we have a pair of monoid homomorphisms

$$\mathrm{End}_\alpha(E) \hookrightarrow \mathrm{End}_\alpha(S) \twoheadrightarrow \mathrm{End}_\alpha(E) \qquad \text{(II.1)}$$

The first is given by $h \mapsto \kappa(h, S)$, while the second is given by $r \mapsto \alpha(r)$. Because their composite is the identity, it follows that the first is an injection and the second is a surjection. In particular, $\mathrm{End}_\alpha(E)$ is identified with a submonoid of $\mathrm{End}(S)$.

Definition II.2 *Let $E = \alpha(S)$ be the elements of S. The* anonymity monoid *$Anon(k) = Anon_\alpha(k)$ of $k : S \to N$ is the submonoid of* $\mathrm{End}_\alpha(E)$ *consisting of all $h \in \mathrm{End}_\alpha(E)$ satisfying $k \circ \kappa(h, S) = k$.*

Remarks:
1. The intuitive justification for this definition is as follows. By Kerckhoffs' assumption, an attacker has complete knowledge of S, N, and k. This knowledge together with observations of the state of N is the basis for anything the attacker may deduce about the state of S. If $k = k \circ \kappa(h, S)$, then the same deductions must apply after the system has been transformed by h, so these deductions cannot discriminate among elements equated by h. One may even think of the deduction itself as transforming by h. For instance, if an anonymous transformation h exchanges elements a and b, then any deduction about a available to the attacker must equally well apply to b and *vice*

versa. All elements thus equated have "dissolved into a crowd."

It should be emphasized that anonymity and secrecy are distinct notions. Anonymity within a group does not preclude deductions about its members. Indeed, this would defy common sense. What it does preclude is making deductions about individuals that do not apply to all members of the group equally. For example, suppose that it is apparent in a voting system that a vote has been unanimous. In this case, an attacker can obviously deduce every voter's choice (the same choice). From a non-unanimous vote, the attacker can deduce nothing about the vote of any individual voter. While the two concepts have an affinity, it is an advantage to be able both to isolate them and to experiment with their interactions, as is possible in our framework.

2. Anonymity in the context of split cofibrations admits a great simplification from the general case. First, by lemma A.4, the knowledge morphism k factors uniquely as $k = u' \circ \kappa(f, S)$ where $f = \alpha(k)$. Now suppose that $h \in \mathrm{End}_\alpha(E)$. Then $h \in Anon(k)$ exactly when the following sequence of equivalent equations hold:

$$\begin{aligned} k &= k \circ \kappa(h, P) \\ u' \circ \kappa(f, S) &= u' \circ \kappa(f, S) \circ \kappa(h, P) \\ u' \circ \kappa(f, S) &= u' \circ \kappa(f \circ h, S) \\ f &= f \circ h \end{aligned}$$

It follows that the vertical part u' of k has no bearing on anonymity, so *only relabelling operations are relevant to anonymity in split cofibrations.* Moreover, the symmetry equation $k = k \circ \kappa(h, P)$ collapses to the equation $f = f \circ h$ in the base category $\mathcal{B}$, which can be expected to be much simpler than the process category. We summarize these facts in the following proposition.

Proposition II.3*Let $f = \alpha(k)$. Then $Anon(k) = Anon(\kappa(f, S))$. Moreover, if $h \in \mathrm{End}_\alpha(E)$, then $h \in Anon(k)$ exactly when $f = f \circ h$.*

3. For any category $\mathcal{P}$, the identity functor $\mathrm{id} : \mathcal{P} \to \mathcal{P}$ is a split cofibration. In this situation, the anonymity monoid $Anon_{\mathrm{id}}(k)$ is the submonoid of $\mathrm{End}(S)$ consisting of those $h \in \mathrm{End}(S)$ for which $k \circ h = k$. This is the universal situation, in the sense that every other cofibration factors through this one trivially, and there are natural monoid homomorphisms

$$Anon_\alpha(k) \hookrightarrow Anon_{\mathrm{id}}(k) \twoheadrightarrow Anon_\alpha(k) \qquad \text{(II.2)}$$

obtained from II.1.

4. Suppose k is the identity morphism $k : S \to S$. This amounts to saying that the attacker has complete knowledge of the system. In this case, $Anon(k)$ is the trivial monoid. Indeed, this is clear if $\alpha = \mathrm{id}$ and from this the general case follows by II.2. The same conclusion holds if k is a monomorphism. Consequently, there is no anonymity if the attacker has complete knowledge.

5. Suppose t is a terminal object of $\mathcal{P}$ (roughly speaking, a process that does nothing), and k is the unique morphism $k : S \to t$. This amounts to saying that the attacker has no knowledge at all. In this situation, every transformation of S is anonymous: $k \circ \kappa(h, S) = k$ is automatic, there being only one morphism to t. Consequently, there is complete anonymity if the attacker has no knowledge.

6. It is intuitively clear that the greater the attacker's knowledge, the less the anonymity. The previous two remarks illustrate the extreme positions on this spectrum. Suppose now that $k : S \to N$ factors as

$$\begin{tikzcd} S \arrow[r, "k'"'] \arrow[rr, bend left, "k"] & N' \arrow[r, "v"'] & N \end{tikzcd}$$

This amounts to saying that k' carries more information than k. Here, we view S as fixed, but the attacker's knowledge as varying. In this case, we have an injection $Anon(k') \hookrightarrow Anon(k)$, so the anonymity does indeed decrease with k'.

7. Suppose again that $k : S \to N$ factors, but now view the attacker's knowledge as fixed. We write the factorization

$$\begin{tikzcd} S \arrow[r, "u"'] \arrow[rr, bend left, "k"] & S' \arrow[r, "k'"'] & N \end{tikzcd}$$

to emphasize this viewpoint. Here, the attacker can deduce more information about S' than about S. However, a comparison between $Anon(k')$ and $Anon(k)$ is complicated by the fact that these two live in different places, namely $\mathrm{End}_\alpha(E')$ and $\mathrm{End}_\alpha(E)$. In general, there is no natural map between these in either direction, but there is a relation $R = \mathrm{End}_\alpha(f) \subseteq \mathrm{End}_\alpha(E) \times \mathrm{End}_\alpha(E')$ defined as follows. Write $E = \alpha(S)$ and $E' = \alpha(S')$ and let $f : E \to E'$ be $f = \alpha(u)$. Then $R = \{(h, h') \in \mathrm{End}_\alpha(E) \times \mathrm{End}_\alpha(E') : h' \circ f = f \circ h\}$.

$$\begin{tikzcd} E \arrow[r, "f"] \arrow[d, "h"'] & E' \arrow[d, "h'"] \\ E \arrow[r, "f"] & E' \end{tikzcd}$$

This relation is in fact a monoid relation, which means by definition that
1.$(1,1) \in R$
2.$(h_1, h_1'), (h_2, h_2') \in R \implies (h_1h_2, h_1'h_2') \in R$
If f is monomorphic, this relation is a partial function from $\mathrm{End}_\alpha(E')$ to $\mathrm{End}_\alpha(E)$ while if it is an epimorphism it is a partial function from $\mathrm{End}_\alpha(E)$ to $\mathrm{End}_\alpha(E')$.
By proposition II.3, there is no loss of generality in assuming that $u = \kappa(f, S)$, from which it follows that $S' = f_*(S)$. For instance, if $\mathcal{P}$ is some semantics of a process algebra such as CCS, then S' could be the result of renaming or hiding events in S.

Lemma II.4*Suppose $u = \kappa(f, S)$. Then $R(Anon(k')) \subseteq Anon(k)$.*

Proof: Let $h \in R(Anon(k'))$. Then there is some $h' \in Anon(k')$ for which $(h, h') \in R$. By definition, $h' \circ f = f \circ h$, and $h_*(S) = S$. We have

$$\begin{aligned} k \circ \kappa(h, S) &= k' \circ \kappa(f, S) \circ \kappa(h, S) \\ &= k' \circ \kappa(f \circ h, S) \\ &= k' \circ \kappa(h' \circ f, S) \\ &= k' \circ \kappa(h', S') \circ \kappa(f, S) \\ &= k' \circ \kappa(f, S) \\ &= k \end{aligned}$$

■

Corollary II.5*Suppose $u = \kappa(f, S)$ with f epimorphic. If $(h, h') \in \mathrm{End}_\alpha(f)$ then $h \in Anon(k)$ if and only if $h' \in Anon(k')$.*

Proof: This follows from the lemma, because if f is epimorphic, then R is a partial function from $\mathrm{End}_\alpha(E)$ to $\mathrm{End}_\alpha(E')$. ■
By the above lemma, identifying an anonymous transformation in the narrowed system S' can determine anonymous transformations in the system S. This can be useful for simplifying computations. A typical use would be in analyzing a process with an alphabet Σ for which we are primarily concerned about anonymity among a subset $A \subseteq \Sigma$. Rather than directly studying the anonymity of k, which may involve unwanted complications, we could study that of the k' corresponding to an f which hides all events of $\Sigma \setminus A$.
8. In the context of anonymity, the distinction between cofibration and the dual notion of fibration is significant. While cocartesian liftings express relabellings, which relate to observation of a system, cartesian liftings express restrictions, which relate to controlling a system. We hope to incorporate fibrations into a more comprehensive theory of active attacks, involving both knowledge and interactions.

III. Hoare Trace Languages as a Cofibered Category

In this section we will recall a simple and familiar model of concurrency called *Hoare trace languages*. In order to fit it into our framework, we review how it can be given the structure of a cofibered category. This model can provide semantics for Hoare's CSP process algebra. This section therefore provides definitions of anonymity applicable to every system specified in CSP. In the following section, we will illustrate this with explicit CSP examples.

If $\mathcal{A}$ is a set, denote by $\mathcal{A}^*$ the set of all finite strings of elements of $\mathcal{A}$. If s and t are two strings, denote their concatenation by st. A prefix of a string u is a string s for which $u = st$ for some string t. A set of strings is called *prefix closed* if it contains every prefix of each of its elements.

Definition III.1 *A* Hoare trace language *(HTL)* *is a pair* $(S, \mathcal{A})$ *in which* $\mathcal{A}$ *is a set called the* alphabet *and* S *is a prefix closed subset* $S \subseteq \mathcal{A}^*$.

HTLs can serve as a model concurrency. The alphabet consists of the actions or events in which a process can participate, and the set S of strings is the set of possible initial sequences of events encountered as the process executes.

The set of HTLs with a fixed alphabet $\mathcal{A}$ is partially ordered by inclusion. We may view this partial order as a category. This structure is meaningful in terms of process behavior. If $S \subseteq T \subseteq \mathcal{A}$, then $(T, \mathcal{A})$ is capable of simulating $(S, \mathcal{A})$ – in other words, $(T, \mathcal{A})$ can do anything $(S, \mathcal{A})$ can do.

There is a more general notion of simulation for HTLs. Note that a function $f : \mathcal{A} \to \mathcal{B}$ induces a function $f^* : \mathcal{A}^* \to \mathcal{B}^*$ by substitution: $f^*(\alpha_1 \cdots \alpha_n) = f(\alpha_1) \cdots f(\alpha_n)$. Henceforth, we will drop the superscript and write f for f^*. Define a *simulation* of an HTL $(S, \mathcal{A})$ by $(T, \mathcal{B})$ to be a function $f : \mathcal{A} \to \mathcal{B}$ for which $f(S) \subseteq T$.

We can define a category HTL in which the objects are HTLs and the morphisms are simulations. This category admits a split cofibration (α, κ) over the category Sets of sets. Here α is the forgetful functor $\alpha : \mathsf{HTL} \to \mathsf{Sets}$ which maps an object $(S, \mathcal{A})$ to its underlying alphabet $\mathcal{A}$, and maps a morphism $(S, \mathcal{A}) \to (T, \mathcal{B})$ to its underlying function $f : \mathcal{A} \to \mathcal{B}$. The splitting κ is given as follows. If $f : \mathcal{A} \to \mathcal{B}$ is a function, and $(S, \mathcal{A})$ is an HTL, then $\kappa(f, (S, \mathcal{A}))$ is the simulation given by f from $(S, \mathcal{A})$ to $(f(S), \mathcal{B})$.

We can now state a definition of anonymity for processes modeled by HTL.

Definition III.2 (Anonymity for HTLs) *Let* $k : (S, \mathcal{A}) \to (T, \mathcal{B})$ *be the morphism of* HTL *determined by a function* $K : \mathcal{A} \to \mathcal{B}$. *Then a function* $H : \mathcal{A} \to \mathcal{A}$ *is in* $Anon(k)$ *when*
1. $H(S) = S$
2. $K \circ H = K$

Of course, this definition is nothing but definition II.2 stated explicitly for HTL.

Here, the fibration is over Sets, but we have freedom to use other base categories. One possibility is the category of partial functions, i.e. the category whose objects are sets and whose morphisms are partial functions. It is also sometimes convenient to allow arbitrary relations between sets. Note that while partial functions may be considered as relations, they do not behave as relations in models of concurrency. The category HTL can be enlarged so that it fibers over relations, or over partial functions, but these encompass distinct kinds of expressiveness. If f is a partial function, and $f(a)$ is undefined, then $f(s)$ erases occurrences of a. On the other hand, if the partial function were interpreted as a relation, and $f(a)$ is undefined, then $f(s)$ is empty whenever a occurs in s. We can interpret the partial function behavior to mean that f is really a monoid homomorphism $\mathcal{A}^* \to \mathcal{B}^*$ specified on the generators $\mathcal{A}$. Hence, $f(a)$ undefined really means $f(a) = 1$.

To generalize both partial functions and relations at once, we can use the category of monoid relations. To be precise, this is the category whose objects are sets and in which a morphism $\mathcal{A} \to \mathcal{B}$ is specified by giving a monoid relation (see section II, remark 7) $R \subseteq \mathcal{A}^* \times \mathcal{B}^*$. A relation R is considered to be a monoid relation by equating it with the monoid relation R^* it generates under the inclusion $R \subseteq \mathcal{A} \times \mathcal{B} \subseteq \mathcal{A}^* \times \mathcal{B}^*$. A partial function f is considered to be the monoid relation generated by $\{(a, f(a)) : f(a) \text{ defined}\} \cup \{(a, 1) : f(a) \text{ undefined}\}$. Then HTL can be enlarged in a way that it admits a split cofibration over the category of monoid relations. For this, we take a morphism $(S, \mathcal{A}) \to (T, \mathcal{B})$ to be specified by a monoid relation $R \subseteq \mathcal{A}^* \times \mathcal{B}^*$ satisfying the same condition as for functions, namely $R(S) \subseteq T$. The splitting is also described exactly as before: if $R \subseteq \mathcal{A}^* \times \mathcal{B}^*$ is a monoid relation and $(S, \mathcal{A})$ is an HTL, then $\kappa(f, (S, \mathcal{A}))$ is the simulation given by R from $(S, \mathcal{A})$ to $(R(S), \mathcal{B})$.

The expressiveness of the category of monoid relations comes at a price. In computational terms, some morphisms in this category are unwieldy to specify, even for small label sets. In terms of our view of processes, monoid relations 'violate' the granularity of actions by relating sequences of actions of different lengths, which may or may not be appropriate for a given application. The question of which category to fiber over is a matter of context and judgment.

To restate the definition for anonymity for HTLs for the enlarged category fibered over monoid relations requires only that we replace the word 'function' with the phrase 'monoid relation' in definition III.2.

IV. A Simple Voting Example in CSP

In this section we present a simple voting example as a CSP process and analyze it for anonymity. The basic reference for CSP is [Hoa85], while [Ros98] also covers many subsequent developments. The reader may wish to compare the analysis in this section with [SS96].

The voting system will have N voters, V_1, V_2, ..., V_N, each of whom casts either a yes or a no vote, and then stops. In this example, we treat the yes or no choice as nondeterminism. If we denote by *vote.k.yes* the event of V_k casting a vote of yes, and similarly for a vote of no, the definition of V_k is

$$V_k = (\mathit{vote.k.yes} \to \mathit{STOP}) \sqcap (\mathit{vote.k.no} \to \mathit{STOP})$$

Recall that in CSP the symbol $\sqcap$ denotes nondeterministic choice. The ensemble Voters_N of all the voters is simply the V_i acting concurrently. Recalling that in CSP the symbol $\|$ denotes concurrency, we have

$$\mathit{Voters}_N = V_1 \,\|\, V_2 \,\|\, \cdots \,\|\, V_N$$

The alphabet of events for this process is the set

$$\mathcal{V} = \{\mathit{vote.k.v} : k \in \{1, 2, \ldots, N\}, v \in \{\mathit{yes}, \mathit{no}\}\}$$

We will represent a tally of y yes votes and n no votes as $\mathit{Tally}_{y,n}$. We view $\mathit{Tally}_{y,n}$ as a deterministic process that first announces its tally with an event *tally.y.n* and thereafter responds to any voting event *vote.k.v* by becoming a new tally, appropriately incremented. More precisely,

$$\mathit{Tally}_{y,n} = \mathit{tally.y.n} \to \mathit{Tally}'_{y,n},$$

where $\mathit{Tally}'_{y,n}$ responds to *vote.k.yes* for any k by becoming $\mathit{Tally}_{y+1,n}$, and responds to *vote.k.no* by becoming $\mathit{Tally}_{y,n+1}$. In CSP, this may be denoted

$$\mathit{Tally}'_{y,n} = x : \mathcal{V} \to P(x)$$

where

$$P(\mathit{vote.k.v}) = \begin{cases} \mathit{Tally}_{y+1,n} & \text{if } v = \mathit{yes} \\ \mathit{Tally}_{y,n+1} & \text{if } v = \mathit{no} \end{cases}$$

Finally, the entire voting system is

$$\mathit{Voting}_N = \mathit{Voters}_N \,\|\, \mathit{Tally}_{0,0}$$

Let us pause to be more precise about the alphabets of these processes. The tally events that are required by Voting_N are all the *tally.y.n* for which y and n are nonnegative integers satisfying $y + n \leq N$. Denote the set of such events

$$\mathcal{T} = \{\mathit{tally.y.n} : y, n \in \mathbb{N}, y + n \leq N\}$$

Then we take Voting_N and all the $\mathit{Tally}_{y,n}$ processes to have alphabet $\mathcal{V} \cup \mathcal{T}$. Note that because the alphabet of Voters_N is disjoint from $\mathcal{T}$, Voters_N does not synchronize on tally events.

We now define the attacker's knowledge as a CSP process. Clearly, if the attacker can see the events *vote.k.v*, there can be no anonymity. On the other hand, it is intuitively clear that if these events are not visible, the voters are anonymous. Thus, we define

$$\mathit{Screen}_N = \mathit{Voting}_N \setminus \mathcal{V}$$

Recall that in CSP the symbol $\setminus$ denotes concealment, so Screen_N indeed represents Voting_N with all voting events removed.

A more liberal choice for the attacker's knowledge would be to replace each voting event *vote.k.v* with an event *vote.v* stripped of the reference to the voter. This can be indicated in CSP using change of symbols. Let $\mathcal{V}' = \{\mathit{vote.yes}, \mathit{vote.no}\}$ and define the function $f : \mathcal{V} \to \mathcal{V}'$ by $f(\mathit{vote.k.v}) = \mathit{vote.v}$. Then we set

$$\mathit{Screen}'_N = f(\mathit{Voting}_N)$$

Now, to analyze the anonymity of this system, we employ the traces semantics of CSP, with target in the Hoare trace languages. Thus, interpreting definition II.2 and applying proposition II.3, anonymity in CSP can be summarized as follows.

Definition IV.1 *Given a CSP process System with alphabet $\mathcal{A}$, and a simulation k : System $\to$ Screen, the anonymity monoid is the set of relations $R : \mathcal{A} \to \mathcal{A}$ for which $R(\mathit{System}) = \mathit{System}$ and $k \circ R = k$.*

When we apply this definition to the simulation $\mathit{Voting}_N \to \mathit{Screen}_N$ we find that the transposition of every pair of voters is in the anonymity monoid. More precisely, for every $i, j \in \{1, 2, \ldots, N\}$, the transpositions of *vote.i.yes* with *vote.j.yes* and *vote.i.no* with *vote.j.no* are in the anonymity monoid. Rather than present the easy proof, we will merely illustrate how one particular trace is transformed by a transposition. Take $N = 4$ and let R be the transposition of voters 1 and 2. For example, *tally*.0.0, *vote*.4.*no*, *tally*.0.1, *vote*.1.*yes*, *tally*.1.1, *vote*.2.*yes*, *tally*.2.1 is a valid trace of $Voting_4$. Applying R to it yields: *tally*.0.0, *vote*.4.*no*, *tally*.0.1, *vote*.2.*yes*, *tally*.1.1, *vote*.1.*yes*, *tally*.2.1. Now, this is certainly different from the original trace, but after we apply the simulation by $Screen_4$, the difference disappears; both traces become *tally*.0.0, *tally*.0.1, *tally*.1.1, *tally*.2.1. Similarly, if we apply the simulation by $Screen'_4$, both traces become *tally*.0.0, *vote.no*, *tally*.0.1, *vote.yes*, *tally*.1.1, *vote.yes*, *tally*.2.1. Moreover, since every permutation can be obtained as a succession of transpositions, this implies that every permutation of the voters is also in the anonymity monoid. This confirms our expectation that in the voting, all voters are distinct, but indistinguishable, to the attacker.

Remark IV.2 *The definition of anonymity in [SS96] fits into our framework as follows. The abstraction of events in that paper plays the role of the knowledge relation k. There are three basic kinds of abstraction: hiding, renaming, and masking. Within the category of monoid relations, all three can be represented as $\kappa(f, P)$ for some suitable monoid relation f. Let P denote a CSP process and let ABS denote some composition of abstractions. Let A be a set of events and let h_A be the hiding relation on A, that maps all elements of A to* 1 *and leaves other elements unchanged. Then the definition of Schneider-Sidiropoulos anonymity of P with respect to the set of events A is ${h_A}^{-1}(h_A(ABS(P))) = ABS(P)$.*

Proposition IV.3 *With notation as above, P satisfies Schneider-Sidiropoulos anonymity if and only if the anonymity monoid of h_A on $ABS(P)$ contains ${h_A}^{-1} \circ h_A$.*

V. A Probabilistic Model of Concurrency

In order to show that probability is within the scope of our framework, we need an appropriate category theoretic account of a probabilistic model of concurrency. An important such model is *probabilistic transition systems*. This model is a probabilistic generalization of the familiar labelled transition systems, but where an action leads from a given state to a probability distribution of states. This model also incorporates pure nondeterminism, in that multiple distributions may be associated with the same action at the same state.

Probabilistic transition systems are very convenient as a target of the semantics of probabilistic process algebras. For instance, [JLY01] presents an elegant procedure for extending essentially any non-probabilistic process algebra by introducing a probabilistic internal choice operator. The semantics of this extended algebra is in terms of probabilistic transition systems. In view of this, the account we give here of probabilistic transition systems automatically endows a large class of probabilistic process algebras with a definition of anonymity.

For the sake of simplicity, we will assume that all probability spaces X are countable and discrete, so that probability distributions can be represented as functions $p : X \to [0, 1]$ satisfying $\sum_{x \in X} p(x)$. Denote by $\mathrm{Dist}(X)$ the set of all probability distributions on X.

Definition V.1 *A* probabilistic transition system *(PTS) is a quadruple* (Σ, L, T, Π) *where*
1. Σ *is a nonempty finite set of* states
2. L *is a set of* actions
3. $T \subseteq \Sigma \times L \times \mathrm{Dist}(\Sigma)$ *is the* transition relation
4. $\pi_0 \in \mathrm{Dist}(\Sigma)$ *is the* initial distribution

We have adopted this definition from [JLY01]. Numerous variants of this definition have appeared in the literature, dating all the way back to Rabin's probabilistic automata [Rab63], a deterministic variant which specifies a set of final states.

The interpretation of the PTS structure is as follows. The system initially assumes a state probabilistically based on the initial distribution. Suppose $(s, a, \pi_1) \in T$. Then the action a can cause a transition from state s to another state of Σ according to the probability distribution π. A suggestive notation for the transition relation is to write $s \xrightarrow[T]{\alpha} \pi$, or simply $s \xrightarrow{\alpha} \pi$, when $(s, \alpha, \pi) \in T$.

There may be multiple transitions (s, a, π_1), $(s, a, \pi_2), \ldots \in T$ emanating from the same state s for the same action a. In this case, when the system is in state s, and action a is performed, the choice among the transition distributions $\pi_1, \pi_2, \ldots$ is nondeterministic.

A *simulation* β between PTS's $S = (\Sigma, L, T, \pi_0)$ and $S' = (\Sigma', L', T', \pi_0')$ is a pair $\beta = (f, h)$ where $h : L \to L'$ and $f : \Sigma \to \Sigma'$ are functions satisfying

$$f_* \pi_0 = \pi_0'$$

and

$$s \xrightarrow[T]{a} \pi \quad \Longrightarrow \quad f(s) \xrightarrow[T']{h(a)} f_* \pi.$$

We can generalize f to be a general relation from Σ to Σ'. In [LS91], a definition is given of *probabilistic bisimulation* between *deterministic* probabilistic transition systems involving a relation on states. This definition has been widely adopted and it carries over to the case at hand. For this we need to recall a definition which first appeared in [JL91], indicating how a relation between sets induces a relation between probability distributions on those sets.

Definition V.2 *Let $R \subseteq X \times Y$ be a relation. Define the induced relation $R^* \subseteq \mathrm{Dist}(X) \times \mathrm{Dist}(Y)$ as follows. A pair (p, q) is in R^* exactly when there is a distribution $W(x, y)$ on $X \times Y$ for which $p(x) = \sum_{y \in R(x)} W(x, y)$ and $q(x) = \sum_{x \in R(y)} W(x, y)$ and $W(x, y) = 0$ unless $(x, y) \in R$.*

Definition V.3 *A* simulation *β between PTS's $S = (\Sigma, L, T, \pi_0)$ and $S' = (\Sigma', L', T', \pi_0')$ is a pair $\beta = (f, h)$ where $h : L' \to L$ is a function and $f : \Sigma \to \Sigma'$ is a relation on states such that*
1. $\pi_0' \in f_* \pi_0$
2. Whenever $s \xrightarrow{a} \pi$ is a transition in S and $s' \in f(s)$, then there is a transition $s' \xrightarrow{h(a)} \pi'$ in S'.

It is straight forward to define a category PTS in which the objects are PTS's and the morphisms are simulations. For instance, composition is defined $(f_1, h_1) \circ (f_2, h_2) = (f_1 \circ f_2, h_1 \circ h_2)$. We omit the verification that this composition determines a simulation and satisfies the conditions required of the morphisms of a category.

There is a forgetful functor $\alpha : \mathsf{PTS} \to \mathsf{Sets}$ defined:
1. for an object $S = (\Sigma, L, T, \pi_0)$ of PTS, $\alpha(S) = L$
2. for a morphism $\beta = (f, h)$, $\alpha(\beta) = h$.

Let $S = (\Sigma, L, T, \pi_0)$ be a PTS and let $h : L' \to L$ be a function. Then we define the push-forward PTS to be $h_* S = (\Sigma, L', T', \pi_0)$ where T' is defined to be the collection of all transitions $(s, h(a), \pi)$ for which $(s, a, \pi) \in T$.

There is a natural simulation $S \to h_* S$ given by $\beta = (1, h)$, lifting the morphism h in Sets.

Proposition V.4 *The morphism $\beta : S \to h_* S$ is cocartesian.*

Proof: Suppose $S'' = (\Sigma'', L'', T'', \pi_0'')$ and $\beta'' : S \to S''$ is a morphism with $\beta'' = (f'', h' \circ h)$. We must show that there is a unique morphism $\beta' : h_* S \to S''$ for which $\beta'' = \beta' \circ \beta$ and $\alpha(\beta') = h'$. By the second condition, $\beta' = (g, h')$ for some relation g. The first condition states that $(f'', h' \circ h) = (g, h') \circ (1, h) = (g, h' \circ h)$. Therefore, $\beta' = (f'', h')$ is the only possible morphism satisfying the conditions. To check that it is indeed a morphism, note first that $\pi_0'' \in f_*'' \pi_0$ because this same condition is required of β''. Next, suppose that $s \xrightarrow{h(a)} \pi$ in $h_* S$ corresponding to a transition $s \xrightarrow{a} \pi$ in S, and let $s'' \in f''(s)$. Then because β'' is a morphism, there is some transition $s'' \xrightarrow{h'(h(a))} \pi'$ in S'' with $\pi' \in f_*'' \pi$. This verifies the second condition. ■

Proposition V.5 *For a function $h : L' \to L$ of sets, and a PTS S with set of actions L, set $\kappa(h, S) = (1, h) : S \to h_* S$. Then (α, κ) is a split cofibration of PTS over Sets.*

Acknowledgments

The problem of formalizing anonymity was suggested to the author by Jay Lepreau, who provided support through the Flux Research Group.

Funding was largely provided by the Defense Advanced Research Projects Agency, monitored by the Air Force Research Laboratory, Rome Research Site, USAF, under agreement F30602-99-1-0503.

Appendix

I. Cofibrations

Let $\alpha : \mathcal{P} \to \mathcal{B}$ be a functor. We will say that an object S of $\mathcal{P}$ *lies over* an object A of $\mathcal{B}$ if $\alpha(S) = A$. Similarly, a morphism u of $\mathcal{P}$ *lies over* a morphism f of $\mathcal{B}$ if $\alpha(u) = f$. If A is an object of $\mathcal{B}$, the *fiber* (of α) over A is the subcategory of $\mathcal{P}$ consisting of all objects of $\mathcal{P}$ lying over A and all morphisms of $\mathcal{P}$ lying over id_A. Clearly, every object S of $\mathcal{P}$ is in a unique fiber, namely the fiber over $\alpha(S)$. Denote the fiber over A by $\alpha^{-1}(A)$.

Definition A.1 *Let $f : A \to A'$ be a morphism in $\mathcal{B}$. A cocartesian lifting of f is a morphism $u : P \to P'$ of $\mathcal{P}$ lying over f and satisfying the following universal condition: for every morphism $u'' : P \to P''$ for which $\alpha(u'') = f' \circ f$ for some f', there is a unique morphism $u' : P' \to P''$ for which $u'' = u' \circ u$.*

Definition A.2 *The functor α is an cofibration if for every morphism $f : A \to A'$ in $\mathcal{B}$ and for every object P lying over A, there is a cocartesian lifting $u : P \to P'$ of f.*

We now recall the more rigid concept of a split cofibration, which is all that we use in the paper.

A *lifting* for a functor $\alpha : \mathcal{P} \to \mathcal{B}$ is a function that specifies a morphism $\kappa(f, S)$ of $\mathcal{P}$ lying over a given morphism f of $\mathcal{B}$ and with specified source S (necessarily lying over the source of f). In other words, for each morphism $f : A \to A'$ of $\mathcal{B}$, and each object P lying over A, there is given a morphism $\kappa(f, P)$ of $\mathcal{P}$ satisfying $\alpha(\kappa(f, P)) = f$. Let us say that a lifting κ is a *splitting* if it satisfies the following properties:
1. $\kappa(\mathrm{id}_A, P) = \mathrm{id}_P$ for all objects A of $\mathcal{B}$ and objects P lying over A
2. $\kappa(g \circ f, P) = \kappa(g, P') \circ \kappa(f, P)$, where P' is the target of $\kappa(f, P)$.

Definition A.3 *The pair (α, κ) is a split cofibration if and only if each morphism $\kappa(f, P)$ is a cocartesian lifting of f.*

Lemma A.4 *Let κ be a splitting of α. The pair (α, κ) is a split cofibration if every morphism $u : P \to P'$ of $\mathcal{P}$ factors uniquely as*

$$u = u' \circ \kappa(\alpha(u), P)$$

for some morphism u' in the same fiber as P'.

We omit the proof.

If (α, κ) is a split cofibration, then every morphism $f : A \to A'$ in $\mathcal{B}$ induces a functor $f_* : \alpha^{-1}(A) \to \alpha^{-1}(A')$ as follows. On objects, $f_*(P)$ is the target of $\kappa(f, P)$. Now suppose $u : P \to P'$ is a morphism in $\alpha^{-1}(A)$. Then the morphism $\kappa(f, P') \circ u$ lies over f. By the lemma, this morphism can be uniquely expressed as $\kappa(f, P') \circ u = u' \circ \kappa(f, P)$. We define $f_*(u) = u'$.

$$\begin{tikzcd}
P \arrow[r, "\kappa(f,P)"] \arrow[d, "u"'] & f_*(P) \arrow[d, "u'=f_*(u)"] \\
P' \arrow[r, "\kappa(f,P')"] & f_*(P')
\end{tikzcd}$$

The above rather streamlined account suffices for our purposes. See [Bén85] for more details.

References

[Bén85] Jean Bénabou, *Fibered categories and the foundations of naive category theory*, Journal of Symbolic Logic **50** (1985), no. 1, 10–37.

[Hoa85] C. A. R. Hoare, *Communicating sequential processes*, Prentice-Hall, Englewood Cliffs, NJ, 1985, & 0-13-153289-8.

[JL91] Bengt Jonsson and Kim Guldstrand Larsen, *Specification and refinement of probabilistic processes*, Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science (Amsterdam, The Netherlands) (Albert R. Meyer, ed.), IEEE Computer Society Press, IEEE Computer Society Press, 15–18 July 1991, pp. 266–277.

[JLY01] Bengt Jonsson, Kim G. Larsen, and Wang Yi, *Probabilistic extensions of process algebras*, Handbook of Process Algebras (J.A. Bergstra, A. Ponse, and S.A. Smolka, eds.), Elsevier, North Holland, 2001.

[LS91] Kim G. Larsen and Arne Skou, *Bisimulation through probabilistic testing*, Information and Computation **94** (1991), no. 1, 1–28.

[Mil89] R. Milner, *Communication and concurrency*, International Series in Computer Science, Prentice Hall, 1989, SU Fisher Research 511/24.

[PW87] Andreas Pfitzmann and Michael Waidner, *Networks without user observability*, Computers & Security **6** (1987), no. 2, 158–166.

[Rab63] M. O. Rabin, *Probabilistic automata*, Information and Control **6** (1963), 230–245.

[Ros98] W. A. Roscoe, *Theory and Practice of Concurrency*, first ed., Prentice-Hall Europe, Hertfordshire, 1998.

[SS96] Schneider and Sidiropoulos, *CSP and anonymity*, ESORICS: European Symposium on Research in Computer Security, LNCS, Springer-Verlag, 1996.

[SS99] Paul F. Syverson and Stuart G. Stubblebine, *Group principals and the formalization of anonymity*, FM'99—Formal Methods, Volume I (Berlin) (Jeanette M. Wing, Jim Woodcock, and Jim Davies, eds.), Lecture Notes in Computer Science, vol. 1708, Springer-Verlag, 1999, pp. 814–833.

[WN95] G. Winskel and M. Nielsen, *Models for concurrency*, Handbook of Logic and the Foundations of Computer Science (S. Abramsky, D. Gabbay, and T. S. E. Maibaum, eds.), vol. 4, Oxford University Press, 1995, pp. 1–148.