A Semiring-based Trace Semantics for Processes with Applications to Information Leakage Analysis

Michele Boreale[1], David Clark[2], and Daniele Gorla[3]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze
[2] Department of Computer Science, King's College London
[3] Dipartimento di Informatica, "Sapienza" Università di Roma

Abstract. We propose a framework for reasoning about program security building on language-theoretic and coalgebraic concepts. The behaviour of a system is viewed as a mapping from traces of high (unobservable) events to low (observable) events: the less the degree of dependency of low events on high traces, the more secure the system. We take the abstract view that low events are drawn from a generic *semiring*, where they can be combined using product and sum operations; throughout the paper, we provide instances of this framework, obtained by concrete instantiations of the underlying semiring. We specify systems via a simple process calculus, whose semantics is given as the unique homomorphism from the calculus into the set of behaviours, i.e. *formal power series*, seen as a final coalgebra. We provide a compositional semantics for the calculus in terms of rational operators on formal power series and show that the final and the compositional semantics coincide.

1 Introduction

Security analysis of programs has traditionally been centered on a notion of *non-interference* [16]. Research has mostly been into a functional interpretation whereby a program is acceptable if low-confidentiality variables or actions do not depend on high-confidentiality ones. This approach has been developed in both imperative [24] and process algebraic [15] settings. Non-interference is now generally recognised as enforcing too strict a policy. For this reason, more flexible variants of this concept are often considered. In *declassification*, a program may be declared as acceptable if information can flow from *high* to *low* but only in prescribed ways [12, 25]. In more recent years, attempts have been made to provide methods to *quantify* the amount of leaked information, mostly building on information-theoretic or probabilistic tools [13, 14, 9, 6]. Then a program may be declared as acceptable if the information it leaks does not exceed a prescribed threshold.

In this paper, we propose a framework for reasoning about information leakage that builds on language-theoretic and coalgebraic concepts. The framework offers a unifying view of diverse facets of language security, such as those mentioned above, puts them in a more abstract perspective and possibly paves the way to their unification. It also elucidates interesting connections between language-based security, coalgebras and language theory.

Let us introduce a scenario that motivates our approach. Consider a discrete-time, nondeterministic system P. During the execution of P, certain events, such as updates of high-variables, are under the control of a secret scheduler and not directly observable from the outside. Some other events are observable, including updates of low variables, input/output actions, certain file accesses, and so on. These observable events are not directly controlled by the secret scheduler, and may obey nondeterministic or probabilistic laws. An attacker can perform observations upon the system only at prescribed times, e.g. only upon termination. Moreover, he can have the system re-execute as many times as he wishes: through these repeated executions, we assume the policy of the secret scheduler (high behaviour) remains fixed, while all the possibilities arising from the nondeterministic or probabilistic low behaviour of the system are observed. Through this process, the attacker collects a set of observations $o_1, o_2, ...$ and combines them into a global observation to make deductions about the non-observable events – in essence, about the choices of the secret scheduler. One can think of basically two ways the observations can be combined. The first one is a form of sequential composition, say $\star$, by which a sequence of consecutive observations, e.g. o_1, o_2, o_3, results into a combined observation, $o = o_1 \star o_2 \star o_3$. Note that, from the point of view of the attacker, only the final, combined observation o may be available, not the intermediate o_i – the $\star$ operation may not be actually available to him. The second operation, call it $+$, can be used to combine observations arising from the repeated executions of the system, e.g. $o_1 \star o_2$ and $o_3 \star o_4$, into a global observation $(o_1 \star o_2) + (o_3 \star o_4)$. This operation is therefore available to the attacker. In the end, to each sequence of unobservable events, say π, there corresponds a global observation o, thus defining a mapping from high traces to observations that we name $\mathcal{L}(P)$. This mapping can be deduced from P's specification, which must be assumed to be public. Hence, using $\mathcal{L}(P)$ and the global observation o, the attacker can learn information about the secret sequence π: at least, he can get to know that $\pi \in (\mathcal{L}(P))^{-1}(o)$.

To make a concrete case, consider a system P, informally specified as follows. Either of two unobservable events, h or h', is initially executed, the choice depending on the secret scheduler. Then, h leads to a state where the low-event l is always executed, while h' leads to a state where either of two branches is taken: in the first branch, l' and then l'' are executed, while in the second just l is executed. In any case, the system then terminates. The two branches are taken, respectively, with probability $\frac{3}{4}$ and $\frac{1}{4}$. In this case, the observations o are probability sub-distributions on low-traces, while $\star$ and $+$ are, respectively, the product and sum of sub-distributions (seen as weighted languages). The above specification hence yields $\mathcal{L}(P)(h) = [l \mapsto 1]$ and $\mathcal{L}(P)(h') = [l'l'' \mapsto \frac{3}{4}, l \mapsto \frac{1}{4}]$.

From the point of view of a designer that must assess the security of the system, the mapping $\mathcal{L}(P)$ is the central object of interest. For example, if $\mathcal{L}(P)$ is a constant, then the observed low-event does not depend on the secret sequence of high-events: the system is perfectly secure (see [26] for a similar notion of security, formulated in a synchronous setting, *Nondeducbility on Strategies*). If this is not the case, the designer might at least be interested in learning how many equivalence classes the domain $\mathcal{L}(P)$ is partitioned into (that is the number of pre-images $(\mathcal{L}(P))^{-1}(o)$, for o ranging over the observations): the fewer, the better. Also, he might want to perform quantitative

measures, in case probabilistic behaviour is involved. In the example above, $\mathcal{L}(P)$ can be seen as a stochastic matrix whose rows and columns are indexed by high- and low-traces, respectively, and its capacity can be computed by standard techniques. Indeed, an information theorist might recognize in this example an instance of the noisy *Z-channel* having $\{h, h'\}$ and $\{l,\ l'l''\}$ as an input and an output alphabet, respectively.

In essence, it is crucial for the designer to be able to specify $\mathcal{L}(P)$, generate it and reason on it - e.g. prove that two system specifications generate the same behaviours - in a *compositional*, *syntax-driven* fashion. We face these issues and draw on language-theoretic concepts. We take the general view that observable events are elements of a *semiring* [18], $\mathbb{S}$, whose product and sum correspond to the $\star$ and + operations mentioned above. A set of unobservable, high-events H is assumed. The security significant behaviour of the system, $\mathcal{L}(P)$, is then a mapping from H^* to $\mathbb{S}$, that is a *formal power series* (FPS) on H and $\mathbb{S}$ [18]. We provide a simple process calculus to specify systems, equipped with an operational semantics given in terms of Moore automata. Then, following [23], we characterize the semantic mapping $\mathcal{L}(\cdot)$ in terms of the unique homomorphism from this calculus into the set of formal power series seen as a final coalgebra. We next provide a compositional semantics of the calculus in terms of rational operators on FPS's, defined via *behavioural differential equations* (BDE's) [23]. We show that the final and the compositional semantics coincide. A consequence of this result is a Kleene theorem saying that, in our calculus, all and only the rational FPS's are definable. The benefits of the two semantics can be summed up as follows: the final semantics allows for reasoning – proving equivalences – on systems by co-induction, while the compositional semantics, and in particular the BDE's, can be used for step-wise, syntax-driven generation of the behaviours $\mathcal{L}(P)$, for any P. Throughout the paper, we provide instances of this framework obtained by concrete instantiations of the semiring $\mathbb{S}$, and examples that illustrate these ideas.

The rest of the paper is organized as follows: In Section 2 we provide background notions about semirings and formal power series and introduce a few concrete instances of them that are relevant to information leakage analysis. In Section 3 we give the syntax and operational semantics of the language. In Section 4 we describe the abstract semantics using finality and characterize the semantic mapping in terms of language equivalence. Following this, we provide a compositional semantics and show that the final and the compositional semantics coincide in Section 5. In Section 6, we provide two non-trivial examples illustrating the use of the compositional semantics and of the language as a modelling tool. To round off the paper, in Section 7 we briefly discuss an extension of the language with a simple form of parallel composition. Finally we offer some comparison with related work and directions for future research. All proofs have been confined to an extended version available online [8].

2 Semirings and formal power series

Recall that a *semiring* $\mathbb{S}$ is a tuple $(S, +, \times, 0, 1)$ such that $(S, +, 0)$ is a commutative monoid, $(S, \times, 1)$ is a monoid, $\times$ distributes over + both on the left and on the right, and 0 annihilates both on the left and on the right (i.e., $0 \times o = o \times 0 = 0$ for each $o \in S$). We let $o, o', ...$ range over S. Moreover, given $o_1, \ldots, o_n \in S$, we let $\sum_{i=1\ldots n} o_i$ denote

$o_1 + \ldots + o_n$. A *semiring (endo)morphism* is a function $f : \mathbb{S} \to \mathbb{S}$ such that: $f(0) = 0$, $f(1) = 1$, and for each $o, o' \in \mathbb{S}$, $f(o + o') = f(o) + f(o')$ and $f(o \times o') = f(o) \times f(o')$.

The simplest possible semiring is $\mathbb{B}$, obtained by taking $S = \{0, 1\}$ and $+$ and $\times$ to be the sum and product of booleans, that is *or* and *and*. Other examples of semirings are the natural numbers $\mathbb{N}$ and the nonnegative reals $\mathbb{R}^+$. Every ring, hence every field, is of course a semiring. As an example of a non-commutative semiring, consider a finite and non-empty alphabet A; then, $\mathbb{L} = (2^{A^*}, \cup, \cdot, \emptyset, \{\epsilon\})$, with $\cup$ being language union, $\cdot$ being language concatenation and ϵ being the empty string, is the semiring of languages over A.

Fix a semiring $\mathbb{S} = (S, +, \times, 0, 1)$ and a finite, non-empty alphabet A. A *formal power series* (FPS) over A with coefficients in $\mathbb{S}$ is a function $\sigma : A^* \to \mathbb{S}$. The set of all such functions will be denoted by $\mathbb{F}_{A,\mathbb{S}}$, or simply by $\mathbb{F}$ when no ambiguity arises. Given $\sigma, \tau \in \mathbb{F}$, the sum $\sigma + \tau$ and convolution product $\sigma \times \tau$ are the FPS's defined in the expected manner, that is, by setting for each $w \in A^*$

$$(\sigma + \tau)(w) = \sigma(w) + \tau(w) \qquad (\sigma \times \tau)(w) = \sum_{u,v:uv=w} \sigma(u) \times \tau(v) \tag{1}$$

where, on the right-hand side $+$ ($\sum$) and $\times$ respectively denote sum and product in $\mathbb{S}$. Note that there is no harm in overloading the symbols $+$ and $\times$ as we do here. Indeed, $\mathbb{S}$ can be seen as a subset of $\mathbb{F}$ by identifying each $o \in \mathbb{S}$ with the FPS σ such that $\sigma(\epsilon) = o$ and $\sigma(w) = 0$ elsewhere. This identification is easily seen to preserve the meaning of $+$, $\times$, 0 and 1. It is readily checked that $(\mathbb{F}, +, \times, 0, 1)$ is in turn a (non-commutative) semiring.

Let us now fix a finite, non-empty alphabet L, ranged over by $l, l', \ldots$. In the rest of the paper, elements of L will usually be interpreted as observable, *low confidentiality actions*, as opposed to unobservable, high confidentiality actions, to be introduced in the next section. For the time being, however, there is no need to fix a specific interpretation of L. We let $\lambda, \lambda', \ldots$ range over L^*. The semiring $\mathbb{WL}$ of *weighted (low-)traces* is defined as $\mathbb{F}_{L,\mathbb{R}^+}$. That is, weighted (low-)traces are functions $o : L^* \to \mathbb{R}^+$, with operations of sum and product defined as in (1) above. The reason for our interest in this semiring is that it includes all functions $o : L^* \to [0, 1]$ such that $\sum_{\lambda \in L^*} o(\lambda) = 1$, that is, all *probability distributions* on low traces, as well as all functions o such that $\sum_{\lambda \in L^*} o(\lambda) \leq 1$, that is, all *probability sub-distributions*. Note that neither of these two sets forms a semiring, which explains why it is mathematically convenient to work with the larger set $\mathbb{WL}$. In what follows we shall sometimes take the freedom of writing down weighted (low-)traces as formal sums. For instance, $\frac{1}{3}ll' + \frac{2}{3}ll''$ denotes the element $o \in \mathbb{WL}$ such that: $o(\lambda) = \frac{1}{3}$ if $\lambda = ll'$, $o(\lambda) = \frac{2}{3}$ if $\lambda = ll''$ and $o(\lambda) = 0$ for any other $\lambda \in L^*$.

Let us give another instance of (noncommutative) semiring related to security analysis. Given any non-empty set V of program variables, a *store* is a partial function $m : V \to \mathbb{D}$, where $\mathbb{D}$ is some data-type. Let M be the set of all such stores. Each element of 2^M, the powerset of M, can be thought of as the result of the execution of a nondeterministic program. It is natural to endow 2^M with a semiring structure as follows. Let us denote by $m \star m'$ the sequential composition of two stores, defined thus: $(m \star m')(v) = m'(v)$ if $m'(v)$ is defined, $(m \star m')(v) = m(v)$ if $m(v)$ is defined and $m'(v)$ undefined, $(m \star m')(v)$ is undefined if neither of $m(v)$, $m'(v)$ are defined. In other words,

$m \star m'$ describes the effect of running two programs one after another, the first producing m and the second producing m'. Now consider $\mathbb{M} = (2^M, \cup, \star, \emptyset, \{\emptyset\})$, where: $\star$ is extended point-wise to 2^M (that is, given $I_1, I_2 \subseteq M$, $I_1 \star I_2 = \{m \star m' | m \in I_1, m' \in I_2\}$) and $\emptyset$ denotes the empty set, which is also the nowhere defined partial function. It is readily checked that $\mathbb{M}$ is a semiring.

3 A process calculus

Let us fix a finite, non-empty alphabet H, ranged over by $h, h', \ldots$. It is convenient to think of H as a set of unobservable, *high-confidentiality actions* (as opposed to the set L introduced in the preceding section; the two sets are assumed to be disjoint). We let $\pi, \pi', \ldots$ range over H^*. Let us fix a semiring $\mathbb{S}$. The set of all *processes* is given by the following syntax

$$P ::= o \mid h \mid P + P \mid P; P \mid P\langle f \rangle \mid P^*$$

where $o \in \mathbb{S}$, $h \in H$ and $f : \mathbb{S} \to \mathbb{S}$ is a semiring morphism. As usual, $+$, $;$ and $*$ denote nondeterministic choice, sequential composition and iteration, respectively; $P\langle f \rangle$ is a filtering operator that applies the filter f – a morphism on the semiring – to the observable events produced by P; the condition that f be a morphism appears to be quite natural, and yields a compositional way to compute filter applications. Given processes $P_1, \ldots, P_n$, we let $\sum_{i=1\ldots n} P_i$ denote $P_1 + \ldots + P_n$, where the summands are arranged in any arbitrary fixed order. By convention, we let this summation denote $0 \in \mathbb{S}$ when $n = 0$. In what follows, we shall not commit to any specific semiring, even though our reference instance is meant to be $\mathbb{WL}$. The set of all processes is denoted by $\mathcal{P}_{\mathbb{S}}$, or simply by $\mathcal{P}$ when there is no need to be specific about $\mathbb{S}$.

A *measure*, $\Delta : \mathcal{P}_{\mathbb{S}} \to \mathbb{S}$, is a map from processes to the semiring $\mathbb{S}$. Let $\mathcal{M}$ be the set of all measures. For any $P \in \mathcal{P}$, we let $\delta(P)$ denote the measure Δ s.t. $\Delta(Q) = 1$ if $Q = P$, $\Delta(Q) = 0$ otherwise; note that here $0, 1 \in \mathbb{S}$. It is useful to define operations of internal sum and scalar product for measures. For each P:

$$(\Delta + \Delta')(P) \triangleq \Delta(P) + \Delta'(P) \qquad (o \times \Delta)(P) \triangleq o \times \Delta(P) \tag{2}$$

where on the right hand side of the definitions the operations are those of the semiring $\mathbb{S}$. Such an overload of the symbols $+$ ($\sum$) and $\times$ is harmless, as any ambiguity is easily resolved by the context. With these operations, every measure can be written as $\Delta = \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(P)$. A few syntactic operations on measures will be useful. Syntactic right-multiplication by a process: if $\Delta = \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(P)$, then $\Delta; Q \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(P; Q)$. Syntactic left-multiplication, $Q; \Delta$, is defined similarly. Finally, syntactic filtering: with the same Δ as above, $\Delta\langle f \rangle \triangleq \sum_{P \in \mathcal{P}} f(\Delta(P)) \times \delta(P\langle f \rangle)$.

When describing the semantics, the following two notable measures will turn out to be useful. For every $P \in \mathcal{P}$:

$$0_{\mathcal{M}}(P) \triangleq 0 \qquad 1_{\mathcal{M}}(P) \triangleq \begin{cases} 1 & \text{if } P = 1 \\ 0 & \text{otherwise} \end{cases}$$

The operational semantics of $\mathcal{P}$ is given by a pair of functions $(\mathrm{w}, \longrightarrow)$. Here, for each P, $\mathrm{w}(P) \in \mathbb{S}$ is the *final weight* of P, corresponding to the observation that can be

made upon P in the current state. A non-zero weight may be understood as indicating the possibility of immediate termination. Specifically, $\mathrm{w} \in \mathcal{M}$ is a measure defined by induction on P as follows[1]:

$$\begin{array}{rrr} \mathrm{w}(o) = o & \mathrm{w}(h) = 0 & \mathrm{w}(P_1; P_2) = \mathrm{w}(P_1) \times \mathrm{w}(P_2) \\ \mathrm{w}(P\langle f \rangle) = f(\mathrm{w}(P)) & & \mathrm{w}(Q^*) = 1 \quad \text{if } \mathrm{w}(Q) = 0 \\ \mathrm{w}(P_1 + P_2) = \mathrm{w}(P_1) + \mathrm{w}(P_2) & & \mathrm{w}(Q^*) = 0 \quad \text{if } \mathrm{w}(Q) \neq 0 \end{array}$$

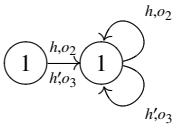
The function $\longrightarrow : (\mathcal{P} \times H) \to \mathcal{M}$, describes the effect of executing a high action and making a transition to a measure. As customary, $(P, h, \Delta) \in \longrightarrow$ will be written as $P \xrightarrow{h} \Delta$. The judgments defining $P \xrightarrow{h} \Delta$ are reported below, where we assume $h' \neq h$.

$$h \xrightarrow{h} 1_{\mathcal{M}} \qquad h' \xrightarrow{h} 0_{\mathcal{M}} \qquad o \xrightarrow{h} 0_{\mathcal{M}} \qquad \frac{P \xrightarrow{h} \Delta_1 \qquad Q \xrightarrow{h} \Delta_2}{P + Q \xrightarrow{h} \Delta_1 + \Delta_2}$$

$$\frac{P \xrightarrow{h} \Delta_1 \qquad Q \xrightarrow{h} \Delta_2}{P; Q \xrightarrow{h} (\Delta_1; Q) + (\mathrm{w}(P) \times \Delta_2)} \qquad \frac{P \xrightarrow{h} \Delta}{P\langle f \rangle \xrightarrow{h} \Delta\langle f \rangle} \qquad \frac{P \xrightarrow{h} \Delta}{P^* \xrightarrow{h} \Delta; P^*}$$

The rules should be self explanatory. In particular, the rule for sequential composition states that the h-derivative of $P; Q$ results from summing up h-derivatives originating from P, with Q as a sequel, and from Q; the latter contributes to the sum *only if* P may terminate immediately. The rule for filtering P with f applies the filter f to every element of the derivative of P. The rule for P^* is obvious if one thinks that Kleene's law, namely $P^* = 1 + P; P^*$, should remain valid in our setting.

The operational semantics $(\mathrm{w}, \longrightarrow)$ can be turned into a more traditional representation in terms of state-transition machines. Recall that a *weighted automaton* [18, 23] is like a nondeterministic automaton, but both its arcs and its states are also labelled with *weights* taken from a semiring. Here, we define a weighted automaton where states are $\mathcal{P}$, the state labeling function is $\mathrm{w}(\cdot)$ and the transition relation $\longrightarrow \subseteq \mathcal{P} \times H \times \mathbb{S} \times \mathcal{P}$ is defined thus: $(P, h, o, P') \in \longrightarrow$, written $P \xrightarrow{h,o} P'$, whenever $P \xrightarrow{h} \Delta$ and $\Delta(P') = o \neq 0$. As an example, the weighted automaton for the process $Q = (o_2; h + o_3; h')^*$ is given here on the right, where the leftmost state corresponds to process Q and the rightmost one to $1; Q$. In the next section, we shall introduce an abstract semantics that equates automata with the same weighted language. It will turn out that the classical law $Q = 1; Q$ holds also in our setting; a possible application of such a law could be simplification of the previous automaton to one with just one state (the rightmost one).



To conclude, let us fix $\mathbb{S} = \mathbb{WL}$ and give a specification in our language of the Z-channel mentioned in the Introduction. The input alphabet is $h, h' \in H$ and the output

[1] Note that the semantics of Q^* is usually taken as undefined when $\mathrm{w}(Q) \neq 0$: the reason is evident if one tries to expand Q^* according to Kleene's law, namely $Q^* = 1 + Q; Q^*$. Here, for simplicity, in case $\mathrm{w}(Q) \neq 0$ we stipulate $\mathrm{w}(Q^*) = 0$, so as to avoid dealing with a partial semantic function.

alphabet is $l, l'l'' \in L^*$; let $p \in [0,1]$. Then

$$Z = h; l + h'; pl'l'' + h'; (1-p)l\,.$$

As we shall see, this turns out to be equivalent to $h; l + h'; (pl'l'' + (1-p)l)$.

4 Abstract semantics

We first describe the abstract semantics of $\mathcal{P}$ by finality and then characterize the semantic mapping in terms of (weighted) language equivalence.

We endow $\mathcal{P}$ with a Moore automaton structure[2] and then define its semantics coalgebraically, following [23, 7]. Recall that a *Moore automaton* with inputs in a finite non-empty alphabet A and outputs in K is a triple (Q, δ, γ) where Q is a (not necessarily finite) set of states, $\delta : Q \times A \to Q$ is a transition function and $\gamma : Q \to K$ is an output function. Let us keep A and K fixed. Central to this treatment is the notion of bisimulation.

Definition 1 (bisimulation). *Given $M = (Q, \delta, o)$, a* bisimulation *is a binary relation $\mathcal{R} \subseteq Q \times Q$ such that, whenever $(q, q') \in \mathcal{R}$ then $\gamma(q) = \gamma(q')$ and $(\delta(q,a), \delta(q',a)) \in \mathcal{R}$, for every $a \in A$. We write $q \sim q'$ if there exists a bisimulation relating q and q'.*

The relation $\sim$ over Q is easily seen to be an equivalence relation and a bisimulation in turn. A *homomorphism* between two Moore automata M and M' is a function ϕ mapping the states of M to the states of M' such that, with an obvious symbology, for each $q \in Q$, $\gamma(q) = \gamma'(\phi(q))$ and, for each $a \in A$, $\phi(\delta(q,a)) = \delta'(\phi(q), a)$. The class of all Moore automata has a final object $\mathcal{F}$ that can be characterized in terms of FPS's. Specifically, we let $\mathcal{F}$ be the Moore automaton (Q, δ, γ) defined thus:

- $Q = \mathbb{F}_{A,K}$;
- $\delta(\sigma, a) = \sigma_a$, where $\sigma_a(w) = \sigma(aw)$, for each $w \in A^*$;
- $\gamma(\sigma) = \sigma(\epsilon)$.

Theorem 1 (Finality and Coinduction principle [23]). *$\mathcal{F}$ is final in the class of Moore automata with inputs in A and outputs in K. That is, for every such automaton M there exists a* unique *homomorphism $\phi : M \to \mathcal{F}$. Moreover, for every q and q' states of M, it holds that $q \sim q'$ if and only if $\phi(s) = \phi(s')$.*

We proceed now to endow $\mathcal{P}$ with a Moore automaton structure, with inputs in H and outputs in the semiring $\mathbb{S}$. Then, the above results will give us: (1) a notion of bisimulation, and (2) a canonical way of interpreting processes as FPS's, which is fully abstract w.r.t. bisimilarity. The construction goes as follows. We extend the weight function and transition relation to $\mathcal{M}$ by linearity. That is, if we let $\Delta_{P,h}$ be the unique measure such that $P \xrightarrow{h} \Delta_{P,h}$ (for each P, h and Δ), then we have:

[2] To be precise, we are endowing $\mathcal{M}$ with a Moore automaton structure, i.e. we are considering Moore automata whose states are measures. With some abuse of terminology, we can consider states as processes, once we see a process P as the Dirac's measure $\delta(P)$.

– $\mathrm{w}(\Delta) \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \mathrm{w}(P)$;
– $\Delta \overset{h}{\longmapsto} \Delta_h$, where $\Delta_h \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \Delta_{P,h}$.

Now, we let $\mathcal{A} \triangleq (\mathcal{M}, \delta, \mathrm{w})$, where $\delta(\Delta, h) = \Delta_h$: this is a Moore automaton with inputs in H and outputs in $\mathbb{S}$. Observe that $\mathcal{P}$ is naturally embedded in $\mathcal{M}$, once one identifies P with the measure $\delta(P)$. We now let $P \sim Q$ stand for $\delta(P) \sim \delta(Q)$. It is crucial for the compositionality of the semantics that bisimilarity over $\mathcal{P}$ be a congruence.

Theorem 2. *For every $P, Q, R \in \mathcal{P}$ such that $P \sim Q$ and for every semiring morphism $f : \mathbb{S} \to \mathbb{S}$, it holds that:*

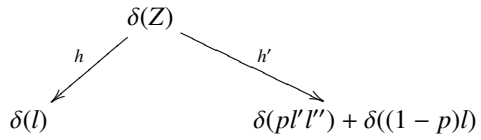
$$(1)\ P + R \sim Q + R \qquad (2)\ P;R \sim Q;R \qquad (3)\ P\langle f \rangle \sim Q\langle f \rangle \qquad (4)\ P^* \sim Q^*$$

Let us denote by $\mathcal{L}$ the unique homomorphism from $\mathcal{A}$ to $\mathcal{F}$ given by Theorem 1; it is a function of type $\mathcal{M} \longrightarrow \mathbb{F}$, mapping every measure to a FPS. We want now to give a more explicit characterization of this homomorphism in terms of the operational semantics $(\mathrm{w}, \longrightarrow)$ of $\mathcal{P}$. To this purpose, we extend the notion of h-derivative of a state Δ, previously written Δ_h, to sequences of high actions $\pi \in H^*$ in the expected way: $\Delta_\epsilon \triangleq \Delta$ and $\Delta_{h\pi} \triangleq (\Delta_h)_\pi$.

Proposition 1. *For every Δ and π, $\mathcal{L}(\Delta)(\pi) = \mathrm{w}(\Delta_\pi)$, for every $\pi \in H^*$.*

To conclude, we can define the language generated by a process P, written $\mathcal{L}(P)$, as expected: $\mathcal{L}(P) \triangleq \mathcal{L}(\delta(P))$.

Let us now illustrate the semantics just introduced by a small, concrete example. Let us consider the Z-channel again, $Z = h; l + h'; pl'l'' + h'; (1-p)l$. The Moore automaton generated by $\delta(Z)$ (or, more formally, the portion of the infinite automaton $\mathcal{A}$ that is reachable from $\delta(Z)$) according to the operational rules is given by



So $\mathrm{w}(\Delta_h) = l$, while $\mathrm{w}(\Delta_{h'}) = pl'l'' + (1-p)l$, as expected. The same result is obtained starting from $Z' = h; l + h'; (pl'l'' + (1-p)l)$; thus, $Z \sim Z'$.

5 A compositional construction

We want to provide now another, more informative way of describing the semantic mapping $\mathcal{L}$ discussed in Section 4. In particular, we want to introduce the analog of the process operators over $\mathbb{F}$ and then prove that $\mathcal{L}$ is compositional w.r.t. these process operators (see Corollary 1 below). We follow the approach in [23, 7] and define operators on FPS's via *behavioural differential equations* (BDE's). Generally speaking, a BDE is a coinductive specification of a FPS, providing its initial value – $\sigma(\epsilon)$ – and the form of its derivatives σ_h, for every $h \in H$. Of course, one has in general to prove that the given

Initial condition	Condition on derivatives
$o(\epsilon) \triangleq o$	$(o)_h \triangleq 0_{\mathbb{F}}$
$h(\epsilon) \triangleq 0$	$(h)_{h'} \triangleq \begin{cases} 1_{\mathbb{F}} & \text{if } h = h' \\ 0_{\mathbb{F}} & \text{otherwise} \end{cases}$
$(\sigma + \sigma')(\epsilon) \triangleq \sigma(\epsilon) + \sigma'(\epsilon)$	$(\sigma + \sigma')_h \triangleq \sigma_h + \sigma'_h$
$(\sigma; \sigma')(\epsilon) \triangleq \sigma(\epsilon) \times \sigma'(\epsilon)$	$(\sigma; \sigma')_h \triangleq \sigma_h; \sigma' + \sigma(\epsilon) \times \sigma'_h$
$(\sigma\langle f \rangle)(\epsilon) \triangleq f(\sigma(\epsilon))$	$(\sigma\langle f \rangle)_h \triangleq (\sigma_h)\langle f \rangle$
$(\sigma^*)(\epsilon) \triangleq \begin{cases} 1 & \text{if } \sigma(\epsilon) = 0 \\ 0 & \text{otherwise} \end{cases}$	$(\sigma^*)_h \triangleq \sigma_h; \sigma^*$

Table 1. Behavioural Differential Equations (BDE's)

equations have a unique solution. The advantage of this kind of definitions, over explicit but possibly more involved ones, is that they allow for coinductive, step-by-step reasoning on the FPS's they define. The BDE's defining the operators associated to the constructs of the language are given in Table 1. There, for every $\pi \in H^*$, we let

$$0_{\mathbb{F}}(\pi) \triangleq 0 \qquad 1_{\mathbb{F}}(\pi) \triangleq \begin{cases} 1 & \text{if } \pi = \epsilon \\ 0 & \text{otherwise} \end{cases}$$

Indeed, some of these BDE's give rise to operators well-known in the literature on rational series: $\sigma + \sigma'$ and $\sigma; \sigma'$ are, respectively, just the sum and convolution product defined by (1) – so another notation for $\sigma; \sigma'$ is just $\sigma \times \sigma'$, while σ^* is standard iteration (see e.g. [23]). The main result of this section is Corollary 1 below.

Theorem 3. *In $\mathbb{F}$, there exist unique constants 'o' and 'h' and operators '+', ';', '$\langle f \rangle$' and '*' that satisfy the* BDE*'s in Table 1.*

Corollary 1 (compositionality). *In $\mathbb{F}$, the unique constants 'o' and 'h' and operators '+', ';', '$\langle f \rangle$' and '*' defined by the* BDE*'s in Table 1 also satisfy the following equalities:*

$$\mathcal{L}(o) = o \qquad \mathcal{L}(h) = h \qquad \mathcal{L}(P + Q) = \mathcal{L}(P) + \mathcal{L}(Q)$$

$$\mathcal{L}(P\langle f \rangle) = (\mathcal{L}(P))\langle f \rangle \qquad \mathcal{L}(P; Q) = \mathcal{L}(P); \mathcal{L}(Q) \qquad \mathcal{L}(P^*) = (\mathcal{L}(P))^*$$

An obvious consequence of the above result is a Kleene theorem for our language. Recall that a FPS $\sigma \in \mathbb{F}$ is *rational* [18] if it can be inductively built starting from the FPS' o and h ($o \in \mathbb{S}, h \in H$) and using the sum, concatenation (sequential composition) and iteration operators defined above. The result entails that one can always eliminate $(\cdot)\langle f \rangle$, essentially by replacing each o occurring in the scope of $(\cdot)\langle f \rangle$ by $f(o)$.

Proposition 2 (a Kleene theorem). *Let σ be a* FPS*. Then σ is rational if and only if $\sigma = \mathcal{L}(P)$ for some process $P \in \mathcal{P}$.*

6 Examples

6.1 Modeling a "Single Bid" Auction

We model a scenario where each of a certain number of users (three, for simplicity) bids for an item at auction. Each user submits a single (secret) bid to a trusted central server

that, in turn, decides the winner by choosing the user whose bid has the highest value. Let U_1, U_2, U_3 be the users; every user knows his bid and the outcome of the auction produced by the server; the problem is measuring the information that every user has about the other users' bids.

We choose a user, U_1, model his view of the auction and try to understand what inferences he can perform about other users' bids – that is, U_1 represents here the (passive) attacker. U_1's bid (a natural number between 1 and m) is an observable event modeled by actions $l_1, \ldots, l_m$; also the outcome of the auction (i.e., the index of the user that wins the auction) is an observable event modeled by actions l'_1, l'_2, l'_3. On the contrary, the bids of U_2 (taken from $\{1, \ldots, n\}$ and modeled by high actions $h_1, \ldots, h_n$) and of U_3 (taken from $\{1, \ldots, q\}$ and modeled by high actions $h'_1, \ldots, h'_q$) are unobservable events, from U_1's point of view. Let us fix the semiring as $\mathbb{S} = \mathbb{WL}$.

A simple way to model the auction is by the following process:

$$\sum_{j=1}^{n} h_j; \left(\sum_{k=1}^{q} h'_k; \left(\sum_{i=1}^{m} [l_i \mapsto \Pr(l_i)]; o_{i,j,k} \right) \right) \tag{3}$$

where $\Pr(l_i)$ denotes the probability of the event l_i and the element $o_{i,j,k} \in \mathbb{WL}$ determines who is the winner of the auction. The actual definition of $o_{i,j,k}$ depends on how we decide to resolve conflicts arising from different users submitting the same bid. A simple but crude way is to resolve the conflict deterministically, e.g. by choosing the user with lowest index:

$$o_{i,j,k} \triangleq \begin{cases} [l'_1 \mapsto 1] & \text{if } i \geq j \text{ and } i \geq k \\ [l'_2 \mapsto 1] & \text{if } j > i \text{ and } j \geq k \\ [l'_3 \mapsto 1] & \text{otherwise.} \end{cases} \tag{4}$$

A fairer way of choosing the winner is by letting

$$o_{i,j,k} \triangleq \left[l'_t \mapsto \frac{1}{|T_{i,j,k}|} \right]_{t \in T_{i,j,k}} \tag{5}$$

where $T_{i,j,k}$ is the set of user indexes (i.e., $T_{i,j,k} \subseteq \{1, 2, 3\}$) containing the indexes of the users who made the greatest bids among i, j, k. For example, if $i = j = k$, then $T_{i,j,k} = \{1, 2, 3\}$; if $i = j > k$, then $T_{i,j,k} = \{1, 2\}$; if $i > j$ and $i > k$, then $T_{i,j,k} = \{1\}$; and so on.

We let P and Q be the process (3) that uses (4) and (5), respectively, as a definition of $o_{i,j,k}$.

Let us now describe the matrix $\mathcal{L}(P)$. By the BDE's (or the operational semantics), the only entries with non-zero values are $\mathcal{L}(P)(h_j h'_k)(l_i l'_t)$ for $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$, $k \in \{1, \ldots, q\}$ and t such that $o_{i,j,k} = [l'_t \mapsto 1]$; moreover, we have that $\mathcal{L}(P)(h_j h'_k)(l_i l'_t) = \Pr(l_i)$. Suppose now that an a priori probability distribution on high traces, $\Pr(\pi)$, reflecting the bidding behaviour of the users, is publicly known. U_1 can then perform some Bayesan inference about the bids of the other users: these inferences are of the form $\Pr(h_j h'_k \mid l_i l'_t)$; by noting that $\mathcal{L}(P)(h_j h'_k)(l_i l'_t)$ corresponds to $\Pr(l_i l'_t \mid h_j h'_k)$ and by elementary probability theory

$$\Pr(h_j h'_k \mid l_i l'_t) = \frac{\Pr(l_i l'_t \mid h_j h'_k) \cdot \Pr(h_j h'_k)}{\Pr(l_i l'_t)} = \frac{\Pr(l_i) \cdot \Pr(h_j h'_k)}{\Pr(l'_t \mid l_i) \cdot \Pr(l_i)} = \frac{\Pr(h_j h'_k)}{\Pr(l'_t \mid l_i)} .$$

To make a concrete case, let us assume that each user has only two possible bidding values; thus, $m = n = q = 2$. In this case, $\mathcal{L}(P)$ is

	$l_1l'_1$	$l_1l'_2$	$l_1l'_3$	$l_2l'_1$
$h_1h'_1$	$\Pr(l_1)$	0	0	$\Pr(l_2)$
$h_1h'_2$	0	0	$\Pr(l_1)$	$\Pr(l_2)$
$h_2h'_1$	0	$\Pr(l_1)$	0	$\Pr(l_2)$
$h_2h'_2$	0	$\Pr(l_1)$	0	$\Pr(l_2)$

Thus, $\Pr(h_1h'_1 \mid l_1l'_1) = 1$, since $\Pr(l'_1 \mid l_1) = \Pr(h_1h'_1)$: indeed, by (4), the only possibility for U_1 to be the winner if he has bid 1 is to have all the bids at 1. The case for $\Pr(h_1h'_2 \mid l_1l'_3)$ is similar. Let us consider now $\Pr(h_2h'_k \mid l_1l'_2)$, for any $k \in \{1, 2\}$; in this case, $\Pr(l'_2 \mid l_1) = \Pr(h_2)$ because, if U_1 has bid 1 and the winner is U_2, it must be that U_2's bid is 2, no matter of U_3's bid. Thus, $\Pr(h_2h'_k \mid l_1l'_2) = \Pr(h'_k)$, once we assume that the users bids are pairwise independent. Finally, let us consider $\Pr(h_jh'_k \mid l_2l'_1)$, for any j and k. In this case, U_1 will always win; thus, $\Pr(l'_1 \mid l_2) = \Pr(l_2)$ and, hence, $\Pr(h_jh'_k \mid l_2l'_1) = \frac{\Pr(h_jh'_k)}{\Pr(l_2)}$. To sum up:

1. if U_1 bids 1,
   (a) he can determine with certainty the other bids if the winner is himself or U_3: in the first case, the bids are 1 for everybody; in the second case, U_2 has bid 1 and U_3 has bid 2.
   (b) if the winner is U_2, his only uncertainty is on U_3's bid, since he knows that U_2 has bid 2.
2. if U_1 bids 2, he surely wins, but he cannot determine with certainty any other bid.

Let us now see how the matrix changes by passing from P to Q, and thus compare the two implementations of the auction system from the security point of view. The matrix for Q is:

	$l_1l'_1$	$l_1l'_2$	$l_1l'_3$	$l_2l'_1$	$l_2l'_2$	$l_2l'_3$
$h_1h'_1$	$\frac{\Pr(l_1)}{3}$	$\frac{\Pr(l_1)}{3}$	$\frac{\Pr(l_1)}{3}$	$\Pr(l_2)$	0	0
$h_1h'_2$	0	0	$\Pr(l_1)$	$\frac{\Pr(l_2)}{2}$	0	$\frac{\Pr(l_2)}{2}$
$h_2h'_1$	0	$\Pr(l_1)$	0	$\frac{\Pr(l_2)}{2}$	$\frac{\Pr(l_2)}{2}$	0
$h_2h'_2$	0	$\frac{\Pr(l_1)}{2}$	$\frac{\Pr(l_1)}{2}$	$\frac{\Pr(l_2)}{3}$	$\frac{\Pr(l_2)}{3}$	$\frac{\Pr(l_2)}{3}$

As expected, this system has more possible high-traces associated to the same low traces, that now are taken from a larger set. Therefore, in this second implementation of the auction system, U_1 can infer less information about the others' bids; in other words, Q is more secure than P. This statement can be made precise by saying that the capacity (see e.g. [9]) of $\mathcal{L}(Q)$ is less than the capacity of $\mathcal{L}(P)$.

We omit the detailed computation for lack of space. It is worth remarking that all the matrices shown can be calculated in a coinductive way via the BDE's presented in the previous section. Moreover, as discussed in [23], such calculations are mechanizable.

6.2 Imperative computations

This section provides a different way of writing examples; indeed, instead of adopting a process algebraic flavour (like, e.g., in section 6.1), we adopt here a more imperative flavour, by exploiting the semiring of stores, $\mathbb{M}$, described in Section 2. We let $\mu, \mu', ...$ range over sets of stores, i.e. partial functions from a set of variables V to a data domain $\mathbb{D}$ that are both non-empty. Notationally, we write the singleton store $\{[x \mapsto v]\}$ as $[x = v]$.

The filter operator $(\cdot)\langle f\rangle$ can be used to express variable updates and conditionals mostly like in an imperative setting. Indeed, variable updates can be modelled by using elements of the semiring as process actions, like in e.g. $[x = 1]; P$. However, this feature only allows us to assign constants to variables. If we want to copy one variable into another, like in e.g. $x := y$, this trick does not work, and we have to use filters. For example, if $x, y \in V$, then the imperative program fragment $P; x := 0; y := x + 1; Q$ corresponds to the following term in the calculus

$$(P; [x = 0])\langle f_{y:=x+1}\rangle; Q$$

where $f_{y:=x+1} : \mathbb{M} \to \mathbb{M}$ is the morphism defined by

$$\begin{aligned} f_{y:=x+1}(\mu) \triangleq \{m \star [y = m(x) + 1] \ : \ m \in \mu \text{ and } m(x) \text{ is defined}\} \\ \cup \{m \in \mu \ : \ m(x) \text{ is not defined}\}\,. \end{aligned}$$

Similarly, the program fragment P; if $(x \neq y)$ then $y := y + 1$ else $z := 1$ corresponds to the term

$$(\ P\langle f_{(x\neq y)}\rangle)\langle f_{y:=y+1}\rangle \ + \ (P\langle f_{(x=y)}\rangle\ )\ \langle f_{z:=1}\rangle\,.$$

Here the function $f_{(x\neq y)}$ filters out the stores not satisfying the condition $x \neq y$, that is

$$\begin{aligned} f_{(x\neq y)}(\mu) \triangleq \{m \in \mu \,|\, m(x), m(y) \text{ are both defined and } m(x) \neq m(y)\} \\ \cup \{m \in \mu \ : \ m(x) \text{ or } m(y) \text{ is not defined}\}\,. \end{aligned}$$

The other filtering functions are defined as expected.

We can use the above ingredients to model the non-interference scenario commonly employed when reasoning on imperative programs. Specifically, let us assume that the set of variables V is partitioned into low and high ones, viz. V_L and V_H. We shall need a filter $(\cdot)\langle f_L\rangle$ that hides from the attacker the high-part of stores and is defined to be $f_L(\mu) \triangleq \{m_{|V_L} \ : \ m \in \mu\}$. In a term like $P\langle f_L\rangle$, assignments to high variables, $[h = v]$, are not directly observable. Rather, in our modelling, it will be convenient to mark the occurrence of each such assignment with a distinct high event: the semantics $\mathcal{L}(P\langle f_L\rangle)$ then takes care of establishing the correct correspondence between sequences of such events and observed stores. As an example, the program fragment $h := 0; l := h$, where $h \in V_H$ and $l \in V_L$, is modelled as

$$Q = \ (\ (h_0; [h = 0])\langle f_{l:=h}\rangle\ )\ \langle f_L\rangle$$

and, as expected, $\mathcal{L}(Q)(h_0) = [l = 0]$.

In this setting, it is quite natural to model, for instance, a PIN checking scenario. A user chooses a 4-digit PIN and then stores it into a high variable h. The attacker chooses a guess for this PIN and stores it into a low variable l. This behaviour is modelled by

$$Choose \triangleq \left(\sum_{i \in \{0,\ldots,9999\}} h_i; [h = i] \right); \left(\sum_{j \in \{0,\ldots,9999\}} [l = j] \right)$$

The PIN-checker then checks h against l and stores the result of the comparison into the low variable r. The whole system is now modelled by:

$$Check \triangleq (Choose \langle f_{h=l} \rangle; [r = ok] + Choose \langle f_{h \neq l} \rangle; [r = no]) \langle f_L \rangle$$

where the filtering functions $f_{h=l}$ and $f_{h \neq l}$ are defined as expected. We could now generate the function $\mathcal{L}(Check)$ via the BDE's and check that it violates non-interference: indeed, $\mathcal{L}(Check)$ maps the trace h_i to the set of stores $\mu_i = \{[l = i, r = ok]\} \cup \{[l = j, r = no] : j \neq i\}$; therefore, for $i \neq j$, we have $\mu_i \neq \mu_j$. We could make the behaviour of the PIN checker more refined, by e.g. combining the two semirings considered in Section 2 and associate probabilities with the choice of the secret and the attacker's guess.

7 Parallelism

The interpretation of parallelism and synchronization is notoriously problematic when probability is involved. On the other hand, if we content ourselves with just weights – indeed in our calculus we never require weights to add up to 1 – parallelism becomes much easier, as studied e.g. by Hillston [17] and other authors doing stochastic process algebra. In fact, is technically easy to extend the language presented in Section 3 with operators that introduce some form of parallelism. The corresponding operational rules mimics those found in process calculi, e.g. CSP [7]. As a further simplification, in the following we shall confine ourselves to a pure interleaving operator, $\|$. We set $\mathrm{w}(P \| Q) = \mathrm{w}(P) \times \mathrm{w}(Q)$ and introduce the new operational rule

$$\frac{P \xrightarrow{h} \Delta \quad Q \xrightarrow{h} \Delta'}{P \| Q \xrightarrow{h} (\Delta \| Q) + (P \| \Delta')}$$

where, as expected, $(\Delta \| Q)$ is the measure that assigns the weight $\Delta(R)$ to any term of the form $R \| Q$, and yields 0 elsewhere ($(P \| \Delta')$ is defined symmetrically). In the final semantics, this corresponds to the *shuffle* operator on FPS defined by the following BDE: $(\sigma \| \tau)(\epsilon) = \sigma(\epsilon) \times \tau(\epsilon)$ and $(\sigma \| \tau)_h = (\sigma_h \| \tau) + (\sigma \| \tau_h)$, for $h \in H$.

As an example, assume $H = \{h, h'\}$ and $L = \{l, l'\}$ and consider $P \triangleq (h; o + h'; o')^*$, for distinct h, h' and o, o'. This process behaves as a noiseless channel that reveals to the attacker the sequence of actions $\pi \in H^*$ is performed by the secret scheduler. Assume now that two other processes work in parallel with P producing distinct observable effects associated with h and h', thus

$$S \triangleq P \| (h; o')^* \| (h'; o)^* .$$

The system S is a quasi-perfect scrambler, that only reveals the total length of the sequence π performed by the three processes. Indeed, assume for instance that $o = [l \mapsto \frac{1}{2}]$ and that $o' = [l' \mapsto \frac{1}{2}]$. Then, in the row π ($\in H^*$) of the matrix $\mathcal{L}(S)$, the probability is uniformly distributed on the low-traces of length k, $\{l, l'\}^k$, where $k = |\pi|$.

8 Concluding Remarks

In the last eight years there has been steady activity in developing concepts, definitions and analyses in the area of measuring information flows for different languages. Ultimately, these aim at being a means of enforcing quantity based security policies. A highly desirable outcome of this effort would be the automatic checking of enforcement via either model checking or program analysis. So far, the efforts have lead to some notable progress for simple imperative languages [14, 22, 21, 4, 11]. By contrast, progress for process algebras has been notably slower. One problem has been establishing appropriate concepts. Lowe's work [19] provided a starting point, developed in quite diverse directions by many authors [6, 9, 10, 20, 3, 1]. Compared with these works, the present paper makes a conceptual, rather than technical, step, by introducing a general, flexible scheme for specifying and analysing regular behaviours of different kinds, of which quantitative ones are just one flavour.

Our study has connections to the work of Rutten and his collaborators on coalgebras. As mentioned throughout the paper, the coalgebraic treatment of streams and FPS's was introduced, in a syntax-free framework, in [23]. In a recent paper [5], they present a systematic way to generate languages of (generalised) regular expressions, and a sound and complete axiomatization thereof, for a wide variety of quantitative systems. There are two major differences between our work and theirs. First, they work with *branching*- rather than *linear*-time semantics: their final coalgebras are not FPS's, but more complicated objects with no natural interpretation in terms of traces, languages and security analysis. Second, they focus on axiomatizations rather than on compositional semantics in terms of rational operators and BDE's, as we do here.

Future developments of the present framework are exploring instantiations and interpretations of the semiring, as well as expanding the process language. Clearly the addition of a parallel operator with synchronization would be a significant enhancement, although it would lead us outside the realm of regular behaviours. So far this extension has presented non-trivial difficulties.

Acknowledgements This work had the benefit of the support of a Royal Society joint international project between King's College, London and the Dipartimento di Informatica, Università "La Sapienza" di Roma. In addition, Clark was supported by the UK EPSRC project EP/C545605/1, Quantified Information Flow.

References

1. A. Aldini and A. Di Pierro. A quantitative approach to noninterference for probabilistic systems. *ENTC*, 99, 2004.

2. A. Askarov and A. Sabelfeld. Tight enforcement of information-release policies for dynamic languages. Proc. of *IEEE CSF*, 2009.
3. M. Backes. Quantifying probabilistic information flow in computational reactive systems. Proc. of ESORICS, volume LNCS 3679. Springer, 2005.
4. M. Backes, B. Kopf, and A. Ribalchenko. Automatic discovery and quantification of information leaks. In *IEEE Symposium on security and privacy*, 2009.
5. F. Bonchi, M. M. Bonsangue, J. J. M. M. Rutten, and A. Silva. Deriving syntax and axioms for quantitative regular behaviours. Proc. of *CONCUR*, volume 5710 of *LNCS*, pages 146–162. Springer, 2009.
6. M. Boreale. Quantifying information leakage in process calculi. *Information and Computation*, 207(6):699–725, 2009.
7. M. Boreale and F. Gadducci. Processes as formal power series: A coinductive approach to denotational semantics. *Theoretical Computer Science*, 360(1-3):440–458, 2006.
8. M. Boreale, D. Clark, and D. Gorla. A Semiring-based Trace Semantics for Processes with Applications to Information Leakage Analysis. Extended version of the present paper. Available from `http://www.dsi.uniroma1.it/~gorla/papers`.
9. K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. Anonymity protocols as noisy channels. Proc. of *TGC*, pages 281–300, 2006.
10. K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. On the bayes risk in information-hiding protocols. *Journal of Computer Security*, 16(5):531–571, 2008.
11. H. Chen and P. Malacaria. Quantitative analysis of leakage for multi-threaded programs. Proc. of *PLAS*, pages 31–40. ACM, 2007.
12. S. Chong and A. C. Myers. Security policies for downgrading. Proc. of *CCS*, pages 189–209, ACM 2004.
13. D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. *ENTCS*, 59, 2002.
14. D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321 – 371, 2007.
15. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5 – 33, 1995.
16. J. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11 – 20, 1982.
17. J. Hillston. A Compositional Approach to Performance Modelling. Cambridge University Press, 1996.
18. W. Kuich and A. Salomaa. Semirings, automata, languages. *Theoretical Computer Science*, 5, 1986.
19. G. Lowe. Quantifying information flow. Proc. of *CSFW*. IEEE, 2002.
20. C. Mu. Measuring information flow in reactive processes. Proc. of *ICICS*. Springer, 2009.
21. C. Mu and D. Clark. An abstraction quantifying information flow over probabilistic semantics. Proc. of *QAPL*. Elsevier, 2009.
22. C. Mu and D. Clark. Quantitative analysis of secure information flow via probabilistic semantics. Proc. of *ARES*. IEEE, 2009.
23. J. J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science*, 308(1-3):1–53, 2003.
24. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selelcted Areas in Communications*, 21(1), 2003.
25. A. Sabelfeld and D. Sands. Dimensions and principles of declassification. Proc. of *CSFW*, pages 255–269, IEEE 2005.
26. J.T. Wittbold, D.M. Johnson. Information flow in nondeterministic systems. Proc. *IEEE Symp. on Security and Privacy*, pages 144-161, IEEE 1990.