Math. Struct. in Comp. Science (2015), *vol.* 25, *pp.* 259–291. © Cambridge University Press 2014
doi:10.1017/S0960129513000601 First published online 10 November 2014

A semiring-based trace semantics for processes with applications to information leakage analysis

MICHELE BOREALE[†], DAVID CLARK[‡] and DANIELE GORLA[§]

[†]*Dipartimento di Statistica, Informatica, Applicazioni – Univ. di Firenze. Viale Morgagni 65, 50134 Firenze, Italy*
Email: boreale@dsi.unifi.it
[‡]*Department of Computer Science, University College London, Gower Street, WC1E 6BT London, United Kingdom*
Email: david.clark@ucl.ac.uk
[§]*Dip. di Informatica – Univ. di Roma 'La Sapienza'. Via Salaria 113, 00198 Roma, Italy*
Email: gorla@di.uniroma1.it

Received 29 November 2011; revised 3 July 2013

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. This compositional, syntax-driven framework lays a foundation for automation and abstraction of a quantified approach to flow security of system specifications.

1. Introduction

Security analysis of programs has traditionally been centred on a notion of *non-interference* (Goguen and Meseguer 1982). 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 (Sabelfeld and Myers 2003) and process algebraic (Focardi and Gorrieri 1995) settings. Non-interference is now generally recognized 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 (Chong and Myers 2004; Sabelfeld and Sands 2005). 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 (Boreale 2009; Chatzikokolakis *et al.* 2008a; Clark *et al.* 2001; Clark *et al.* 2007). Then, a program may be declared as acceptable if the information it leaks does not exceed a prescribed threshold.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

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 make observations of 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: throughout these repeated executions we assume the policy of the secret scheduler (high behaviour) remains fixed, meanwhile (all) the possibilities arising from the nondeterministic or probabilistic low behaviour of the system are observed.

Through the activity described above, the attacker collects a set of observations $o_1, o_2, \ldots$ 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 two basic ways in which the observations can be combined. The first is a form of sequential composition, say $\star$, by which a sequence of consecutive observations, e.g. o_1, o_2, o_3, results in 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 $+$, is a form of disjunction that can be used to combine observations arising from repeated executions of the system, e.g. o and o', into a global observation $o + o'$. 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 minimum, he can discover whether $\pi \in (\mathcal{L}(P))^{-1}(o)$.

As a concrete example, consider a system P, informally specified as follows. Either of two unobservable events, h or h', is executed initially, 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 are probability sub-distributions on low traces, while $\star$ and $+$ are, respectively, the product and sum of distributions. 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 an observed low event does not depend on the secret sequence of high events: the system

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

is perfectly secure (see Wittbold and Johnson (1990) for a similar notion of security, formulated in a synchronous setting and called *Nondeducibility 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 make some flow measurements if 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* (Kuich and Salomaa 1986), $\mathbb{S}$, 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) over the alphabet H with coefficients in $\mathbb{S}$ (Kuich and Salomaa 1986). We provide a simple process calculus to specify systems, equipped with an operational semantics given in terms of Moore automata. Then, following Rutten (2003), we characterize the semantic mapping $\mathcal{L}(\cdot)$ in terms of the unique homomorphism from this calculus into the set of FPS 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) (Rutten 2003). 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 coinduction, while the compositional semantics, and in particular the BDE's, can be used for stepwise, 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 FPS 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 three 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. All the calculations performed in the examples can be carried out manually, as sketched in Section 6, or can be automated, as outlined in Section 8. In Section 9, we offer some comparison with related work and directions for future research. Some technical proofs have been confined to Appendix A. Finally,

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Appendix B contains a synopsis of the meaning and use of the main symbols used throughout the paper.

This is an extended and revised version of Boreale *et al.* (2010), with respect to the preliminary version, this paper contains full proofs, a new application of our framework (namely, the exponentiation attack in Section 6.2), some details on the implementation and a more extensive discussion of related work.

2. Semirings and formal power series

We start by recalling the definition of semirings.

Definition 1. 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 $o \in S$).

If the monoid $(S, \times, 1)$ is commutative, we say $\mathbb{S}$ is a commutative semiring. We let $o, o', \ldots$ range over S. Moreover, given $o_1, \ldots, o_n \in S$, we let $\sum_{i=1\ldots n} o_i$ denote $o_1 + \cdots + o_n$. 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 $f(o \times o') = f(o) \times f(o')$. Whenever necessary or convenient, we shall augment the semiring operation symbols with appropriate subscripts to make the underlying semiring explicit. E.g. we may write $0_{\mathbb{S}}$, $1_{\mathbb{S}}$, $+_{\mathbb{S}}$, $\Sigma_{\mathbb{S}}$ and $\times_{\mathbb{S}}$.

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, $\mathbb{L} = (2^{A^*}, \cup, \cdot, \varnothing, \{\epsilon\})$, 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 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 about A and $\mathbb{S}$. 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 $w \in A^*$

$$(\sigma + \tau)(w) \triangleq \sigma(w) +_{\mathbb{S}} \tau(w) \tag{1}$$

$$(\sigma \times \tau)(w) \triangleq \sum_{u,v\,:uv=w}{}_{\mathbb{S}}\ \sigma(u) \times_{\mathbb{S}} \tau(v). \tag{2}$$

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 $+_{\mathbb{S}}$, $\times_{\mathbb{S}}$, $0_{\mathbb{S}}$ and $1_{\mathbb{S}}$. With this identification, $\mathbb{S} \subseteq \mathbb{F}$. It is readily checked that $(\mathbb{F}, +, \times, 0, 1)$ is in turn a (non-commutative) semiring. In the special case when $|A| = 1$, then $\mathbb{F}_{A,\mathbb{S}}$ can be identified with $\mathbb{S}^\omega$, the set of infinite sequences of elements of $\mathbb{S}$: these are called *streams* in Rutten (2003).

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*,

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

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) and (2) 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 $\sum_{\lambda \in L^*} o(\lambda) \leqslant 1$, 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 $\lambda \in L^*$. Sometimes, we shall write such an o as $[ll' \mapsto \frac{1}{3}, ll'' \mapsto \frac{2}{3}]$.

Let us give another instance of 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 $\mathbb{M} = (2^M, \cup, \star, \varnothing, \{\varnothing\})$, where: $\star$ is extended pointwise 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 $\varnothing$ denotes the empty set, which is also the nowhere defined partial function. It is readily checked that $\mathbb{M}$ is a (non-commutative) semiring. We will let $\mu, \mu', \ldots$ range over $\mathbb{M}$.

Remark. Following the tradition of FPS, throughout the paper several symbols ($+$, $\times$, 0, 1, ...) will be overloaded: such an overloading is harmless, as any ambiguity is easily resolved by the context, e.g. by looking at the type of the operands. For the reader's convenience, we provide in Appendix B a table summarizing all the overloaded symbols: for each symbol, we give its possible uses and say where the corresponding definitions can be found in the paper.

3. A process calculus

3.1. *Syntax.*

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:

$$P ::= o \mid h \mid P + P \mid P ; P \mid P\langle f \rangle \mid P^* \tag{3}$$

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

where $o \in \mathbb{S}$, $h \in H$ and $f : \mathbb{S} \to \mathbb{S}$ is a semiring morphism. As usual, $+$, ; and * denote 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 $P_1 + \cdots + P_n$, where the summands are arranged in any arbitrary fixed order. By convention, we let this summation denote $0_{\mathbb{S}}$ when $n = 0$. Again, the symbols $+$ and Σ are overloaded here, but no ambiguity arises if one looks at the type of the operands.

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 $\mathbb{S}$.

3.2. *Measures*

A *measure* $\Delta : \mathcal{P}_{\mathbb{S}} \to \mathbb{S}$ is a total map from processes to the semiring $\mathbb{S}$ such that its *support* $\mathrm{supp}(\Delta) \triangleq \{P \in \mathcal{P}_{\mathbb{S}} \ : \Delta(P) \neq 0_{\mathbb{S}}\}$ is finite. We denote the set of all measures by $\mathcal{M}$. Intuitively, a measure will be used to describe all possible states (processes) a term can reach by performing a given high action, together with the 'weights' attached to any such transitions. Since the processes we consider are finitely branching, it is quite natural to demand that measures have a finite support.

For any $P \in \mathcal{P}$, we let $\delta(P)$ (Dirac's Delta) denote the measure Δ s.t. $\Delta(P) = 1_{\mathbb{S}}$ and $\Delta(Q) = 0_{\mathbb{S}}$ for any other Q. When describing the semantics, the following two notable measures will turn out to be useful: $1_{\mathcal{M}} \triangleq \delta(1_{\mathbb{S}})$ and $0_{\mathcal{M}}$ such that $0_{\mathcal{M}}(P) \triangleq 0_{\mathbb{S}}$, for every $P \in \mathcal{P}$.

It is useful to define operations of internal sum and scalar product for measures. For each P:

$$(\Delta + \Delta')(P) \triangleq \Delta(P) +_{\mathbb{S}} \Delta'(P) \tag{4}$$

$$(o \times \Delta)(P) \triangleq o \times_{\mathbb{S}} \Delta(P). \tag{5}$$

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: if $\Delta = \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(P)$, then:

$$\Delta; Q \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(P; Q) \tag{6}$$

$$Q; \Delta \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(Q; P) \tag{7}$$

$$\Delta\langle f\rangle \triangleq \sum_{P \in \mathcal{P}} f(\Delta(P)) \times \delta(P\langle f\rangle) \tag{8}$$

where the right hand sides rely on the sum and scalar product defined in (4) and (5).

It is worth remarking the following properties of the operators just defined. Recall that a *semimodule over a semiring* (Kuich and Salomaa 1986) is an algebraic structure satisfying the same axioms as a vector space, with the difference that scalars range in a semiring rather than in a field.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Table 1. *Operational rules.*

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

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

Proposition 1.

1. $\mathcal{M}$ is a semimodule over $\mathbb{S}$ with the internal sum and scalar product defined in (4) and (5);
2. $o \times (\Delta;P) = (o \times \Delta);P$
3. $(\Delta_1;P) + (\Delta_2;P) = (\Delta_1 + \Delta_2);P$
4. $(o \times \Delta)\langle f\rangle = f(o) \times \Delta\langle f\rangle$
5. $(\Delta_1 + \Delta_2)\langle f\rangle = \Delta_1\langle f\rangle + \Delta_2\langle f\rangle$.

Proof. See Appendix A. □

Corollary 1. Proposition 1(1) entails the following laws:

1. $(\Delta_1 + \Delta_2) + \Delta_3 = \Delta_1 + (\Delta_2 + \Delta_3)$.
2. $(o_1 + o_2) \times \Delta = (o_1 \times \Delta) + (o_2 \times \Delta)$.
3. $o \times (\Delta_1 + \Delta_2) = (o \times \Delta_1) + (o \times \Delta_2)$.
4. $o_1 \times (o_2 \times \Delta) = (o_1 \times o_2) \times \Delta$.

3.3. *Operational semantics*

The operational semantics of $\mathcal{P}$ is given by a pair of functions $(\mathrm{w}, \longrightarrow)$. Here, $\mathrm{w} : \mathcal{P}_{\mathbb{S}} \to \mathbb{S}$ assigns each process P its *final weight* $\mathrm{w}(P)$, on the basis of the observation that can be made upon P in the current state. A non-zero weight may be understood as the possibility of observable termination for the current state. Specifically, w is a function defined by induction on P as follows[†]:

$$\begin{array}{lll} \mathrm{w}(o) = o & \mathrm{w}(h) = 0_{\mathbb{S}} & \mathrm{w}(P_1;P_2) = \mathrm{w}(P_1) \times_{\mathbb{S}} \mathrm{w}(P_2) \\ \mathrm{w}(P\langle f\rangle) = f(\mathrm{w}(P)) & \mathrm{w}(Q^*) = 1_{\mathbb{S}} & \mathrm{w}(P_1 + P_2) = \mathrm{w}(P_1) +_{\mathbb{S}} \mathrm{w}(P_2). \end{array} \tag{9}$$

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 in Table 1, where we assume $h' \neq h$. The rules should be self-explanatory. In particular, the rule for sequential composition states

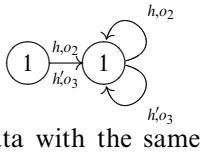
[†] Note that our definition of $\mathrm{w}(Q^*)$ respects some of the traditional laws of Kleene's algebras. For example, $(Q^*)^* = Q^*$ and, whenever $1_{\mathbb{S}}$ is an absorbing element ($1_{\mathbb{S}} +_{\mathbb{S}} o = 1_{\mathbb{S}}$ for each o), also $Q^* = 1 + (Q;Q^*)$. However, the Kleene algebra structure is not needed for our purposes, and we can also consider semirings where $1_{\mathbb{S}}$ is not absorbing.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

that the h-derivative of $P;Q$ (i.e. the measure obtained after the transition from $P;Q$ labelled with h) results from summing up the h-derivatives originating from P, with Q as a continuation 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 of Kleene's law $P^* = 1 + P;P^*$.

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* (Kuich and Salomaa 1986; Rutten 2003) 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 in $\mathcal{P}$, the state labelling 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 the 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 $\{h, h'\} \subseteq H$ and the output alphabet is $\{l, l'l''\} \subseteq L^*$; let $p \in [0, 1]$. Then

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

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[†] and then define its semantics coalgebraically, following Boreale and Gadducci (2006) and Rutten (2003). Recall that a *Moore automaton* with inputs in a finite non-empty alphabet A and outputs in K is a triple (S, t, k) where S is a (not necessarily finite) set of states, $t : S \times A \to S$ is a transition function and $k : S \to K$ is an output function. Let us keep A and K fixed. Central to this treatment is the notion of bisimulation.

Definition 2 (bisimulation). Given $M = (S, t, k)$, a *bisimulation* is a binary relation $\mathcal{R} \subseteq S \times S$ such that, whenever $(s, s') \in \mathcal{R}$, then $k(s) = k(s')$ and $(t(s, a), t(s', a)) \in \mathcal{R}$, for every $a \in A$. We write $s \sim s'$ if there exists a bisimulation relating s and s'.

[†] Formally, 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 processes as states, once we identify a process P with the measure $\delta(P)$.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

The relation $\sim$ over S is easily seen to be an equivalence relation and a bisimulation in turn; we shall call it *bisimilarity*. 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 $s \in S$, $k(s) = k'(\phi(s))$ and, for each $a \in A$, $\phi(t(s,a)) = t'(\phi(s),a)$. 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 (S,t,k) defined thus:

— $S = \mathbb{F}_{A,K}$.
— $t(\sigma,a) = \sigma_a$, where $\sigma_a(w) \triangleq \sigma(aw)$, for each $w \in A^*$ (σ_a is called the *a-derivative* of σ).
— $k(\sigma) = \sigma(\epsilon)$.

Theorem 1 (finality and coinduction principle (Rutten 2003)). $\mathcal{F}$ 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 s and s' states of M, it holds that $s \sim s'$ 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: for each P and h, we let $\Delta_{P,h}$ be the unique measure such that $P \xrightarrow{h} \Delta_{P,h}$; then, we define:

$$\mathrm{w}(\Delta) \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \mathrm{w}(P) \tag{10}$$

$$\Delta \overset{h}{\longmapsto} \Delta_h, \quad \text{where} \quad \Delta_h \triangleq \textstyle\sum_{P \in \mathcal{P}} \Delta(P) \times \Delta_{P,h}. \tag{11}$$

Now, we let $\mathcal{A} \triangleq (\mathcal{M}, t, \mathrm{w})$, where $t(\Delta,h) = \Delta_h$: 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. To prove this result we need two auxiliary lemmata whose proofs are relegated to Appendix A.

Lemma 1. For every $\Delta_1, \Delta_2 \in \mathcal{M}$ such that $\Delta_1 \sim \Delta_2$, it holds that:

1. $\Delta + \Delta_1 \sim \Delta + \Delta_2$, for every $\Delta \in \mathcal{M}$.
2. $o \times \Delta_1 \sim o \times \Delta_2$, for every $o \in \mathbb{S}$.
3. $\Delta_1 ; P \sim \Delta_2 ; Q$, for every $P, Q \in \mathcal{P}$ such that $P \sim Q$.
4. $\Delta_1\langle f \rangle \sim \Delta_2\langle f \rangle$, for every semiring morphism f.

Lemma 2. $\delta(P) + \delta(Q) \sim \delta(P + Q)$.

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:

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

Proof. The first fact is a trivial consequence of Lemmas 1(1) and 2. The second fact is a trivial consequence of Lemma 1(3) and of the fact that, by definition, $\delta(P ; R) = \delta(P) ; R$.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

The third fact is a trivial consequence of Lemma 1(4) and of the fact that, by definition, $\delta(P\langle f\rangle) = (\delta(P))\langle f\rangle$. Let us prove the fourth fact. We prove that

$$\mathcal{R} \triangleq \{(\Delta_1; P^* \ , \ \Delta_2; Q^*) \ : \ P \sim Q \text{ and } \Delta_1 \sim \Delta_2\}$$

is a bisimulation; this trivially implies the result by taking $\Delta_1 = \Delta_2 = 1_{\mathcal{M}}$ and by noting that $1_{\mathcal{M}}; P \triangleq \delta(1_{\mathbb{S}}; P) \sim \delta(P)$, for every P.

— The output is the same:

$$\begin{aligned} \mathrm{w}(\Delta_1; P^*) &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta_1; P^*)(R) \times \mathrm{w}(R) \\ &= \textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times \mathrm{w}(R') \times \mathrm{w}(P^*) \\ &= \mathrm{w}(P^*) \times \mathrm{w}(\Delta_1) \\ &= \mathrm{w}(Q^*) \times \mathrm{w}(\Delta_2) \\ &= \mathrm{w}(\Delta_2; Q^*). \end{aligned}$$

Indeed, $\mathrm{w}(\Delta_1) = \mathrm{w}(\Delta_2)$, since $\Delta_1 \sim \Delta_2$. Moreover, $\mathrm{w}(P^*) = 1_{\mathbb{S}} = \mathrm{w}(Q^*)$.

— The derivatives are still bisimilar:

$$\begin{aligned} (\Delta_1; P^*)_h &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta_1; P^*)(R) \times \Delta_{R,h} \\ &= \textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times \Delta_{R';P^*,h} \\ &= \textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times (\Delta_{R',h}; P^* \ + \ \mathrm{w}(R') \times \Delta_{P^*,h}) \\ &= \textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times \Delta_{R',h}; P^* \\ &\quad + \textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times (\mathrm{w}(R') \times (\Delta_{P,h}; P^*)) \\ &= (\Delta_1)_h; P^* \ + \ (\mathrm{w}(\Delta_1) \times \Delta_{P,h}); P^* \\ &= ((\Delta_1)_h \ + \ \mathrm{w}(\Delta_1) \times \Delta_{P,h}); P^*. \end{aligned}$$

Similarly, for Q in place of P and Δ_2 in place of Δ_1. Now, since $\mathrm{w}(\Delta_1) = \mathrm{w}(\Delta_2)$ and $\Delta_{P,h} \sim \Delta_{Q,h}$, by Lemma 1(2) $\mathrm{w}(\Delta_1) \times \Delta_{P,h} \sim \mathrm{w}(\Delta_2) \times \Delta_{Q,h}$; by $(\Delta_1)_h \sim (\Delta_2)_h$ and Lemma 1(1), $((\Delta_1)_h \ + \ \mathrm{w}(\Delta_1) \times \Delta_{P,h}) \sim ((\Delta_2)_h \ + \ \mathrm{w}(\Delta_2) \times \Delta_{Q,h})$. This suffices to conclude that $((\Delta_1; P^*)_h \ , \ (\Delta_2; Q^*)_h) \in \mathcal{R}$, as desired. □

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 now want to give a more explicit characterization of this homomorphism in terms of the operational semantics $(\mathrm{w}, \longrightarrow)$ of $\mathcal{P}$; this will be useful to carry out explicit calculations (see, e.g. Section 6.2). 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 2. For every Δ and π, $\mathcal{L}(\Delta)(\pi) = \mathrm{w}(\Delta_\pi)$, for every $\pi \in H^*$.

Proof. The proof is by induction on the length of π. For the base case, we have that $\mathcal{L}(\Delta)(\epsilon) = \mathrm{w}(\Delta) \triangleq \mathrm{w}(\Delta_\epsilon)$, where the first equality holds since $\mathcal{L}$ is a homomorphism of Moore automata. For the inductive case, let $\pi \triangleq h\pi'$; then, by the homomorphism properties of $\mathcal{L}$ and induction hypothesis, $\mathcal{L}(\Delta)(h\pi') = \mathcal{L}(\Delta_h)(\pi') = \mathrm{w}((\Delta_h)_{\pi'}) \triangleq \mathrm{w}(\Delta_{h\pi'})$. □

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

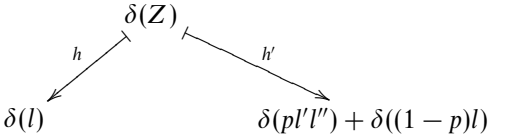
Let us now illustrate the semantics just introduced by a small, concrete example. Let us consider the Z-channel again, $Z = h; l + h'; pl'l'' + h'; (1-p)l$. The Moore automaton

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Table 2. *Behavioural differential equations (*BDE*'s).*

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

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



with $\mathrm{w}(\delta(Z)) = 0_{\mathbb{S}}$, $\mathrm{w}(\Delta_h) = l$ and $\mathrm{w}(\Delta_{h'}) = pl'l'' + (1-p)l$, 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 provide another, more informative way of describing the semantic mapping $\mathcal{L}$ discussed in Section 4. In particular, we want to introduce the analogue of the process operators over $\mathbb{F}$ and then prove that $\mathcal{L}$ is compositional w.r.t. these process operators (see Corollary 2 below). We follow the approach in Boreale and Gadducci (2006) and Rutten (2003) and define operators on FPS's via 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 equations have a unique solution. The advantage of this kind of definition, over explicit but possibly more involved ones, is that it allows for coinductive, step-by-step reasoning on the FPS's it defines. The BDE's defining the operators associated with the constructs of the language are given in Table 2. There, for every $\pi \in H^*$, we let

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

Also note that in the BDE for sequential composition (viz. ';') we treat $\sigma(\epsilon) \in \mathbb{S}$ as a FPS, under the inclusion $\mathbb{S} \subseteq \mathbb{F}$ discussed in Section 2.

Indeed, some of these BDE's give rise to operators which are well-known in the literature on rational series: $\sigma + \sigma'$ and $\sigma;\sigma'$ are, respectively, just the sum and convolution product defined by (1) and (2) – so another notation for $\sigma;\sigma'$ is just $\sigma \times \sigma'$ and we will use these two symbols on FPS's interchangeably, while σ^* is standard iteration (see e.g. Rutten (2003)).

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

As we shall discuss later on (Section 8), these BDE's can be given a direct translation into Haskell's lazy streams and trees, along the lines of Douglas Mcilroy (1999). This provides us an automatic way of calculating $\mathcal{L}(P)$.

The main result of this section is Corollary 2 below. To prove it, we characterize the solutions of the BDE's in an operational manner, as follows. For each $\sigma \in \mathbb{F}$, fix a distinct symbol $\underline{\sigma}$. We consider a syntax of *extended* processes, $\mathcal{EP}$ which includes $\mathcal{P}$, built out of the following grammar which features an extra clause for constants:

$$EP ::= \underline{\sigma} \mid o \mid h \mid EP + EP \mid EP ; EP \mid EP^* . \tag{12}$$

The weight function and operational semantics of Section 3 are extended by the rules

$$\mathrm{w}(\underline{\sigma}) = \sigma(\epsilon) \tag{13}$$

$$\underline{\sigma} \xrightarrow{h} \delta(\underline{\sigma_h}). \tag{14}$$

The definitions and results described for $\mathcal{P}$ – including the Moore automaton definition – carry over to $\mathcal{EP}$. We shall continue to call $\mathcal{L}(\cdot)$ the final morphism mapping $\mathcal{EP}$ to $\mathbb{F}$.

For proving the main result, we need a few auxiliary lemmata.

Lemma 3.
1. $(\mathcal{L}(\Delta))_h = \mathcal{L}(\Delta_h)$;
2. $\Delta \sim \Delta'$ if and only if $\mathcal{L}(\Delta) = \mathcal{L}(\Delta')$.

Proof. Trivial, by definition. □

Lemma 4. $\delta(\underline{\sigma})_h = \delta(\underline{\sigma_h})$.

Proof. $\delta(\underline{\sigma})_h \triangleq \sum_{EP \in \mathcal{EP}} \delta(\underline{\sigma})(EP) \times \Delta_{EP,h} = 1 \times \Delta_{\underline{\sigma},h} = \Delta_{\underline{\sigma},h} = \delta(\underline{\sigma_h})$. □

Lemma 5. $\mathcal{L}(\underline{\sigma}) = \sigma$.

Proof. By induction on the length of π, we show that $\mathcal{L}(\underline{\sigma})(\pi) = \sigma(\pi)$. For the base case, by definition of $\underline{\sigma}$, we have that $\mathcal{L}(\underline{\sigma})(\epsilon) \triangleq \mathcal{L}(\delta(\underline{\sigma}))(\epsilon) \triangleq \mathrm{w}(\delta(\underline{\sigma})) = \sigma(\epsilon)$. For the inductive case, let π be $h\pi'$. By the homomorphism properties of $\mathcal{L}$ and Lemma 4, $\mathcal{L}(\underline{\sigma})(h\pi') \triangleq \mathcal{L}(\delta(\underline{\sigma}))(h\pi') = \mathcal{L}(\delta(\underline{\sigma})_h)(\pi') = \mathcal{L}(\delta(\underline{\sigma_h})(\pi') \triangleq \mathcal{L}(\underline{\sigma_h})(\pi')$; by definition of derivatives in FPS, $\sigma(h\pi') = \sigma_h(\pi')$. We conclude by using the inductive hypothesis. □

Lemma 6. $\delta(\underline{\mathcal{L}(\Delta)}) \sim \Delta$.

Proof. By showing that $\mathcal{R} = \{(\Delta, \delta(\underline{\mathcal{L}(\Delta)}))\}$ is a bisimulation. First, Δ and $\delta(\underline{\mathcal{L}(\Delta)})$ have the same output: indeed, $\mathrm{w}(\delta(\underline{\mathcal{L}(\Delta)})) = \mathrm{w}(\underline{\mathcal{L}(\Delta)}) \triangleq \mathrm{w}(\mathcal{L}(\Delta)(\epsilon) + \sum_{h \in H} h ; \underline{\mathcal{L}(\Delta)_h}) = \mathcal{L}(\Delta)(\epsilon) \triangleq \mathrm{w}(\Delta)$. Second, Δ and $\delta(\underline{\mathcal{L}(\Delta)})$ have the same h-derivative: indeed, $\delta(\underline{\mathcal{L}(\Delta)})_h = \delta(\underline{(\mathcal{L}(\Delta))_h}) = \delta(\underline{\mathcal{L}(\Delta_h)})$, where the first equality is Lemma 4 and the second one follows from Lemma 3(1). This suffices to conclude. □

Lemma 7. $o \times \delta(\underline{\sigma}) \sim \delta(\underline{o \times \sigma})$.

Proof. We prove that $\mathcal{R} \triangleq \{(o \times \delta(\underline{\sigma}), \delta(\underline{o \times \sigma}))\}$ is a bisimulation.

— $\mathrm{w}(o \times \delta(\underline{\sigma})) \triangleq \sum_{EP \in \mathcal{EP}} (o \times \delta(\underline{\sigma}))(EP) \times \mathrm{w}(EP) \triangleq \sum_{EP \in \mathcal{EP}} o \times \delta(\underline{\sigma})(EP) \times \mathrm{w}(EP) = o \times \mathrm{w}(\underline{\sigma}) = o \times \sigma(\epsilon) \triangleq (o \times \sigma)(\epsilon) \triangleq \mathrm{w}(o \times \sigma) = \mathrm{w}(\delta(o \times \sigma))$.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

— $(o \times \delta(\underline{\sigma}))_h \triangleq \sum_{EP \in \mathcal{EP}} (o \times \delta(\underline{\sigma}))(EP) \times \Delta_{EP,h} \triangleq \sum_{EP \in \mathcal{EP}} (o \times \delta(\underline{\sigma})(EP)) \times \Delta_{EP,h} = o \times \Delta_{\underline{\sigma},h} = o \times \delta(\underline{\sigma_h})$. Moreover, $\delta(\underline{o \times \sigma})_h \triangleq \sum_{EP \in \mathcal{EP}} \delta(\underline{o \times \sigma})(EP) \times \Delta_{EP,h} = \Delta_{\underline{o \times \sigma},h} = \delta((o \times \sigma)_h) = \delta(\underline{o \times \sigma_h})$. Indeed, for every $\pi \in H^*$, it holds that $(o \times \sigma)_h(\pi) \triangleq (o \times \sigma)(h\pi) \triangleq o \times \sigma(h\pi) \triangleq o \times \sigma_h(\pi) \triangleq (o \times \sigma_h)(\pi)$. Thus, $((o \times \delta(\underline{\sigma}))_h, \delta(\underline{o \times \sigma})_h) \in \mathcal{R}$, as desired. □

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 2.

Proof. Let us define

$$o \triangleq \mathcal{L}(o) \tag{15}$$
$$h \triangleq \mathcal{L}(h) \tag{16}$$
$$\sigma + \sigma' \triangleq \mathcal{L}(\underline{\sigma} + \underline{\sigma'}) \tag{17}$$
$$\sigma ; \sigma' \triangleq \mathcal{L}(\underline{\sigma} ; \underline{\sigma'}) \tag{18}$$
$$\sigma \langle f \rangle \triangleq \mathcal{L}(\underline{\sigma} \langle f \rangle) \tag{19}$$
$$\sigma^* \triangleq \mathcal{L}(\underline{\sigma}^*). \tag{20}$$

First, let us prove that they satisfy the BDEs in Table 2.

1.— $o(\epsilon) \triangleq \mathcal{L}(o)(\epsilon) \triangleq \mathcal{L}(\delta(o))(\epsilon) = \mathrm{w}(\delta(o)) \triangleq 1 \times \mathrm{w}(o) = o$, where the first equality holds by Proposition 2.

— By Lemma 3(1) and definition of h-derivative for $\delta(o)$, it holds that $(o)_h \triangleq (\mathcal{L}(o))_h \triangleq (\mathcal{L}(\delta(o)))_h = \mathcal{L}(\delta(o)_h) = \mathcal{L}(0_{\mathcal{M}}) = 0_{\mathbb{F}}$.

2.— By Proposition 2, $h(\epsilon) \triangleq \mathcal{L}(h)(\epsilon) \triangleq \mathcal{L}(\delta(h))(\epsilon) = \mathrm{w}(\delta(h)) = \mathrm{w}(h) = 0$.

— By the homomorphism properties of $\mathcal{L}$, $(h)_{h'} \triangleq (\mathcal{L}(h))_{h'} \triangleq (\mathcal{L}(\delta(h)))_{h'} = \mathcal{L}(\delta(h)_{h'})$.

– If $h = h'$, the definition of h-derivative for $\delta(h)$ entails that $\mathcal{L}(\delta(h)_h) = \mathcal{L}(1_{\mathcal{M}}) = 1_{\mathbb{F}}$;

– If $h \neq h'$, the definition of h'-derivative for $\delta(h)$ entails that $\mathcal{L}(\delta(h)_{h'}) = \mathcal{L}(0_{\mathcal{M}}) = 0_{\mathbb{F}}$.

3.— By Proposition 2, $(\sigma + \sigma')(\epsilon) \triangleq \mathcal{L}(\underline{\sigma} + \underline{\sigma'})(\epsilon) \triangleq \mathcal{L}(\delta(\underline{\sigma} + \underline{\sigma'}))(\epsilon) = \mathrm{w}(\delta(\underline{\sigma} + \underline{\sigma'})) = \mathrm{w}(\underline{\sigma} + \underline{\sigma'}) \triangleq \mathrm{w}(\underline{\sigma}) + \mathrm{w}(\underline{\sigma'}) = \mathcal{L}(\underline{\sigma})(\epsilon) + \mathcal{L}(\underline{\sigma'})(\epsilon) = \sigma(\epsilon) + \sigma'(\epsilon)$, where the last equality holds by Lemma 5.

— By (17) and Lemma 3(1), $(\sigma + \sigma')_h \triangleq (\mathcal{L}(\delta(\underline{\sigma} + \underline{\sigma'})))_h = \mathcal{L}(\delta(\underline{\sigma} + \underline{\sigma'})_h)$. By definition of the operational semantics, $\delta(\underline{\sigma} + \underline{\sigma'})_h = \Delta$, where Δ is such that $\underline{\sigma} + \underline{\sigma'} \xrightarrow{h} \Delta$; thus, it must be $\Delta = \Delta' + \Delta''$, where $\underline{\sigma} \xrightarrow{h} \Delta'$ and $\underline{\sigma'} \xrightarrow{h} \Delta''$. By definition of $\underline{\cdot}$, we have that $\Delta' = \delta(\underline{\sigma_h})$ and $\Delta'' = \delta(\underline{\sigma'_h})$. Thus, $\mathcal{L}(\delta(\underline{\sigma} + \underline{\sigma'})_h) = \mathcal{L}(\delta(\underline{\sigma_h}) + \delta(\underline{\sigma'_h})) = \mathcal{L}(\delta(\underline{\sigma_h + \sigma'_h})) \triangleq \mathcal{L}(\underline{\sigma_h + \sigma'_h}) = \sigma_h + \sigma'_h$, where the second equality holds by Lemma 2 and the last one by (17).

4.— By Proposition 2, $(\sigma ; \sigma')(\epsilon) \triangleq \mathcal{L}(\underline{\sigma} ; \underline{\sigma'})(\epsilon) \triangleq \mathcal{L}(\delta(\underline{\sigma} ; \underline{\sigma'}))(\epsilon) = \mathrm{w}(\delta(\underline{\sigma} ; \underline{\sigma'})) = \mathrm{w}(\underline{\sigma} ; \underline{\sigma'}) \triangleq \mathrm{w}(\underline{\sigma}) \times \mathrm{w}(\underline{\sigma'}) = \mathcal{L}(\underline{\sigma})(\epsilon) \times \mathcal{L}(\underline{\sigma'})(\epsilon) = \sigma(\epsilon) \times \sigma'(\epsilon)$, where the last equality holds by Lemma 5.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

— By (18) and Lemma 3(1), $(\sigma;\sigma')_h \triangleq (\mathcal{L}(\delta(\underline{\sigma};\underline{\sigma'})))_h = \mathcal{L}(\delta(\underline{\sigma};\underline{\sigma'})_h)$. By definition of the operational semantics, $\delta(\underline{\sigma};\underline{\sigma'})_h = \Delta$, where Δ is such that $\underline{\sigma};\underline{\sigma'} \xrightarrow{h} \Delta$; thus, it must be $\Delta = (\Delta';\underline{\sigma'}) + (\mathrm{w}(\underline{\sigma}) \times \Delta'')$, where $\underline{\sigma} \xrightarrow{h} \Delta'$ and $\underline{\sigma'} \xrightarrow{h} \Delta''$. By definition of $\underline{\cdot}$, we have that $\Delta' = \delta(\underline{\sigma_h})$ and $\Delta'' = \delta(\underline{\sigma'_h})$. Thus, $\mathcal{L}(\delta(\underline{\sigma};\underline{\sigma'})_h) = \mathcal{L}((\delta(\underline{\sigma_h});\underline{\sigma'})+(\mathrm{w}(\underline{\sigma})\times\delta(\underline{\sigma'_h}))) \triangleq \mathcal{L}(\delta(\underline{\sigma_h};\underline{\sigma'})+(\mathrm{w}(\underline{\sigma})\times\delta(\underline{\sigma'_h})))$. By definition of $\underline{\sigma}$ and by Lemma 7, $\mathrm{w}(\underline{\sigma}) \times \delta(\underline{\sigma'_h}) \sim \delta(\sigma(\epsilon) \times \underline{\sigma'_h})$; moreover, by Lemma 6 and (18), $\delta(\underline{\sigma_h};\underline{\sigma'}) \sim \delta(\mathcal{L}(\delta(\underline{\sigma_h};\underline{\sigma'}))) \triangleq \delta(\underline{\mathcal{L}(\underline{\sigma_h};\underline{\sigma'})}) = \delta(\underline{\sigma_h};\sigma')$. Hence, $\mathcal{L}(\delta(\underline{\sigma_h};\underline{\sigma'}) + (\mathrm{w}(\underline{\sigma}) \times \delta(\underline{\sigma'_h}))) = \mathcal{L}(\underline{\delta(\underline{\sigma_h};\sigma') + \delta(\sigma(\epsilon) \times \underline{\sigma'_h})}) = \mathcal{L}(\delta(\underline{\sigma_h};\sigma' + \underline{\sigma(\epsilon) \times \sigma'_h})) \triangleq \mathcal{L}(\underline{\sigma_h;\sigma' + \sigma(\epsilon) \times \sigma'_h}) = \sigma_h;\sigma' + \sigma(\epsilon) \times \sigma'_h$, where the first equality holds by Lemmas 1(1) and 3(2), the second one by Lemma 2 and the last one by (17).

5.— By Proposition 2, $(\sigma\langle f\rangle)(\epsilon) \triangleq \mathcal{L}(\underline{\sigma}\langle f\rangle)(\epsilon) \triangleq \mathcal{L}(\delta(\underline{\sigma}\langle f\rangle))(\epsilon) = \mathrm{w}(\delta(\underline{\sigma}\langle f\rangle)) = f(\mathrm{w}(\underline{\sigma})) = \sigma(\epsilon)$, where the last equality holds by definition of $\mathrm{w}(\cdot)$ and $\underline{\sigma}$.

— By (19) and Lemma 3(1), $(\sigma\langle f\rangle)_h \triangleq (\mathcal{L}(\delta(\underline{\sigma}\langle f\rangle)))_h = \mathcal{L}(\delta(\underline{\sigma}\langle f\rangle)_h)$. By definition of the operational semantics, $\delta(\underline{\sigma}\langle f\rangle)_h = \Delta$, where Δ is such that $\underline{\sigma}\langle f\rangle \xrightarrow{h} \Delta$; thus, it must be $\Delta = \Delta'\langle f\rangle$, where $\underline{\sigma} \xrightarrow{h} \Delta'$. By definition of $\underline{\cdot}$, we have that $\Delta' = \delta(\underline{\sigma_h})$; thus, $\mathcal{L}(\delta(\underline{\sigma}\langle f\rangle)_h) = \mathcal{L}(\delta(\underline{\sigma_h})\langle f\rangle)$. Now, by the fact that $\delta(EP\langle f\rangle) = (\delta(EP))\langle f\rangle$ and by (19), we have that $\mathcal{L}(\delta(\underline{\sigma_h})\langle f\rangle) = \mathcal{L}(\delta(\underline{\sigma_h\langle f\rangle})) \triangleq \mathcal{L}(\underline{\sigma_h\langle f\rangle}) = \sigma_h\langle f\rangle$.

6.— By Proposition 2, $(\sigma^*)(\epsilon) \triangleq \mathcal{L}(\underline{\sigma^*})(\epsilon) \triangleq \mathcal{L}(\delta(\underline{\sigma^*}))(\epsilon) = \mathrm{w}(\delta(\underline{\sigma^*})) = \mathrm{w}(\underline{\sigma^*}) \triangleq 1_{\mathbb{S}}$.

— By (20) and Lemma 3(1), $(\sigma^*)_h \triangleq (\mathcal{L}(\delta(\underline{\sigma^*})))_h = \mathcal{L}(\delta(\underline{\sigma^*})_h)$. By definition of the operational semantics, $\delta(\underline{\sigma^*})_h = \Delta$, where Δ is such that $\underline{\sigma^*} \xrightarrow{h} \Delta$; thus, it must be $\Delta = \Delta';\underline{\sigma^*}$, where $\underline{\sigma} \xrightarrow{h} \Delta'$. By definition of $\underline{\cdot}$, we have that $\Delta' = \delta(\underline{\sigma_h})$; thus, $\mathcal{L}(\delta(\underline{\sigma^*})_h) = \mathcal{L}(\delta(\underline{\sigma_h});\underline{\sigma^*})$. By Lemma 6 and (20), $\delta(\underline{\sigma^*}) \sim \delta(\underline{\mathcal{L}(\delta(\underline{\sigma^*}))}) \triangleq \delta(\underline{\mathcal{L}(\underline{\sigma^*})}) = \delta(\underline{\sigma^*})$. Thus, $\mathcal{L}(\delta(\underline{\sigma_h});\underline{\sigma^*}) = \mathcal{L}(\delta(\underline{\sigma_h});\underline{\sigma^*}) \triangleq \mathcal{L}(\delta(\underline{\sigma_h};\underline{\sigma^*})) \triangleq \mathcal{L}(\underline{\sigma_h};\underline{\sigma^*}) = \sigma_h;\sigma^*$, where the first equality holds by Lemmas 1(3) and 3(2) and the last one by (18).

Concerning uniqueness, suppose that there is $\hat{o}$ that satisfies the first BDE in Table 2; thus, by the coinduction principle, $o = \hat{o}$. The case for h is similar. Let us suppose that there is '$\hat{+}$' that satisfies the third BDE in Table 2 (unicity of '$;$', '$\langle f\rangle$' and '*' can be proved similarly). We now prove that $(\sigma+\sigma')(\pi) = (\sigma\hat{+}\sigma')(\pi)$, for every π; the proof is by induction on π. For the base case, $(\sigma+\sigma')(\epsilon) = \sigma(\epsilon)+\sigma'(\epsilon) = (\sigma\hat{+}\sigma')(\epsilon)$, since both '$+$' and '$\hat{+}$' satisfy the third BDE in Table 2. For the inductive case, let $\pi = h\pi'$; by definition of h-derivative for FPSs, by the condition on derivatives of the third BDE and by induction hypothesis, $(\sigma + \sigma')(h\pi') = ((\sigma + \sigma')_h)(\pi') = (\sigma_h + \sigma'_h)(\pi') = (\sigma_h\hat{+}\sigma'_h)(\pi') = ((\sigma\hat{+}\sigma')_h)(\pi') = (\sigma\hat{+}\sigma')(h\pi')$. □

Corollary 2 (compositionality). In $\mathbb{F}$, the unique constants 'o' and 'h' and operators '$+$', '$;$', '$\langle f\rangle$' and '*' defined by the BDE's in Table 2 also satisfy the following equalities:

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

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

Proof. The constants defined by (15) and (16) satisfy the first two equalities by definition.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

It suffices to prove that (17) satisfies $\mathcal{L}(P+Q) = \mathcal{L}(P)+\mathcal{L}(Q)$. We know by Lemma 6 that $\delta(P) \sim \delta(\underline{\mathcal{L}(\delta(P))}) \triangleq \delta(\underline{\mathcal{L}(P)})$ and, similarly, that $\delta(Q) \sim \delta(\underline{\mathcal{L}(Q)})$; thus, by Lemmas 1(1) and 2, $\delta(P+Q) = \delta(P)+\delta(Q) \sim \delta(\underline{\mathcal{L}(P)})+\delta(\underline{\mathcal{L}(Q)}) = \delta(\underline{\mathcal{L}(P)}+\underline{\mathcal{L}(Q)})$; thus, by the homomorphism properties of $\mathcal{L}$ and (17), $\mathcal{L}(P+Q) = \mathcal{L}(\underline{\mathcal{L}(P)}+\underline{\mathcal{L}(Q)}) = \mathcal{L}(P)+\mathcal{L}(Q)$.

It suffices to prove that (18) satisfies $\mathcal{L}(P;Q) = \mathcal{L}(P);\mathcal{L}(Q)$. Lemma 6 entails that $\delta(P) \sim \delta(\underline{\mathcal{L}(P)})$ and $\delta(Q) \sim \delta(\underline{\mathcal{L}(Q)})$; thus, by definition of ';' and Lemma 1(3), $\delta(P;Q) \triangleq \delta(P);Q \sim \delta(\underline{\mathcal{L}(P)});\underline{\mathcal{L}(Q)} \triangleq \delta(\underline{\mathcal{L}(P)};\underline{\mathcal{L}(Q)})$. By the homomorphism properties of $\mathcal{L}$ and (18), $\mathcal{L}(P;Q) = \mathcal{L}(\underline{\mathcal{L}(P)};\underline{\mathcal{L}(Q)}) = \mathcal{L}(P);\mathcal{L}(Q)$.

It suffices to prove that (19) satisfies $\mathcal{L}(P\langle f\rangle) = \mathcal{L}(P)\langle f\rangle$. Lemma 6 and Theorem 2(3) entail that $\delta(P\langle f\rangle) \sim \delta(\underline{\mathcal{L}(P)}\langle f\rangle)$; thus, by the homomorphism properties of $\mathcal{L}$ and (19), $\mathcal{L}(P\langle f\rangle) = \mathcal{L}(\underline{\mathcal{L}(P)}\langle f\rangle) = \mathcal{L}(P)\langle f\rangle$.

It suffices to prove that (20) satisfies $\mathcal{L}(P^*) = \mathcal{L}(P)^*$. Lemma 6 and Theorem 2(4) entail that $\delta(P^*) \sim \delta(\underline{\mathcal{L}(P)}^*)$; thus, by the homomorphism properties of $\mathcal{L}$ and (20), $\mathcal{L}(P^*) = \mathcal{L}(\underline{\mathcal{L}(P)}^*) = \mathcal{L}(P)^*$. □

An obvious consequence of the above result is a Kleene theorem for our language. Recall that a FPS $\sigma \in \mathbb{F}$ is *rational* (Kuich and Salomaa 1986) 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 3 (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

We present a few instances of our framework. In each case, after giving an informal description, we provide a formal specification, P, of the system under consideration in our language and then compute $\mathcal{L}(P)$.

In view of the compositional semantics of Section 5 (Theorem 3), the process calculus can be seen as just syntax for FPS's. It is then convenient to work directly in terms FPS's and operators over them. In particular, if $\sigma = \mathcal{L}(P)$, then $\mathcal{L}(P)(w)$ is just $\sigma(w) = \sigma_w(\epsilon)$, for any $w \in H^*$. To compute $\sigma(w)$, assuming that σ is described by a regular expression, we can

1. compute σ_w, the w-derivative of σ. This can be done in a syntax-driven way, by repeatedly applying to σ the equations in the second column of the BDE's table (Table 2) from left to right. This rewrite stops as soon as any h-derivative operator σ_h in the expression is pushed underneath a sequential composition operator (; or equivalently $\times$);
2. compute $\sigma_w(\epsilon)$. Again, this can be done in a syntax-driven way, by the equations in the first column of the BDE's in the same table.

In this section, both these steps will be carried out manually; however, they can be easily automated, as we will show in Section 8.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

6.1. *Modelling 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 an arbitrary 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 modelled 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 modelled by actions l'_1, l'_2, l'_3. On the other hand, the bids of U_2 (taken from $\{1, \ldots, n\}$ and modelled by high actions $h_1, \ldots, h_n$) and of U_3 (taken from $\{1, \ldots, q\}$ and modelled 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:

$$\sum_{j=1}^{n} h_j ; \left(\sum_{k=1}^{q} h'_k ; \left(\sum_{i=1}^{m} [l_i \mapsto \Pr(l_i)] ; o_{i,j,k} \right) \right) \tag{21}$$

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:

$$o_{i,j,k} \triangleq \begin{cases} [l'_1 \mapsto 1] & \text{if} \quad i \geqslant j \quad \text{and} \quad i \geqslant k \\ [l'_2 \mapsto 1] & \text{if} \quad j > i \quad \text{and} \quad j \geqslant k \\ [l'_3 \mapsto 1] & \text{otherwise.} \end{cases} \tag{22}$$

A fairer way of choosing the winner is by letting

$$o_{i,j,k} \triangleq \left[l'_t \mapsto \frac{1}{|T_{i,j,k}|} \right]_{t \in T_{i,j,k}} \tag{23}$$

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

We provide two models of the auction, P and Q, where each is the process (21) using the definitions (22) and (23) respectively to instantiate $o_{i,j,k}$.

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 exists on high traces, $\Pr(\pi)$, reflecting the bidding behaviour of the users and that it is publicly known. U_1 can then perform some Bayesian inference about the bids of the other users: these inferences are

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

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, $m = n = q = 2$. To simplify notation, let us denote with σ_j the second sum in (21) and with σ_{jk} the inner-most one; thus, $\sigma_j = \sum_{k=1,\ldots,q} h'_k ; \sigma_{jk}$. Moreover, notice that $\sigma_{jk} \in \mathbb{WL}$, for every j and k (recall that $\mathbb{WL} = \mathbb{S} \subseteq \mathbb{F}$). With this notation, when $m = n = q = 2$, the process in (21) is written as

$$\sigma = h_1 ; \sigma_1 + h_2 ; \sigma_2 = h_1 ; (h'_1 ; \sigma_{11} + h'_2 ; \sigma_{12}) + h_2 ; (h'_1 ; \sigma_{21} + h'_2 ; \sigma_{22}).$$

By repeatedly applying the BDE's of Table 2 in a syntax-driven fashion, we have that

$$\begin{aligned}
\sigma_{h_1 h'_1} &= (\sigma_{h_1})_{h'_1} \\
&= (\sigma_1)_{h'_1} \\
&= (h'_1)_{h'_1} ; \sigma_{11} + h'_1(\epsilon) \times (\sigma_1)_{h'_1} + (h_2)_{h'_1} ; \sigma_{12} + h_2(\epsilon) \times (\sigma_2)_{h'_1} \\
&= 1_{\mathbb{F}} ; \sigma_{11} + 0_{\mathbb{S}} \times (\sigma_1)_{h'_1} + 0_{\mathbb{F}} ; \sigma_{12} + 0_{\mathbb{S}} \times (\sigma_2)_{h'_1} \\
&= \sigma_{11}
\end{aligned}$$

since

$$\begin{aligned}
\sigma_{h_1} &= (h_1)_{h_1} ; \sigma_1 + h_1(\epsilon) \times (\sigma_1)_{h_1} + (h_2)_{h_1} ; \sigma_2 + h_2(\epsilon) \times (\sigma_2)_{h_1} \\
&= 1_{\mathbb{F}} ; \sigma_1 + 0_{\mathbb{S}} \times (\sigma_1)_{h_1} + 0_{\mathbb{F}} ; \sigma_2 + 0_{\mathbb{S}} \times (\sigma_2)_{h_1} \\
&= \sigma_1\,.
\end{aligned}$$

In the last steps of both the above calculations, we have applied a few simplifying laws: for every FPS τ, it holds that $0_{\mathbb{F}} ; \tau = 0_{\mathbb{S}} \times \tau = 0_{\mathbb{F}}$ and $1_{\mathbb{F}} ; \tau = \tau$. These simplifications are not strictly necessary to complete the computations, but are useful to lighten the notation. Also note that $\sigma_{h_j} = \sigma_j$, $\sigma_{h_j h'_k} = \sigma_{jk}$ and $\sigma_w = 0_{\mathbb{F}}$ for any other w.

When we adopt (22), we have that

$$\begin{aligned}
\sigma_{11} &= [l_1 \mapsto \Pr(l_1)] ; [l'_1 \mapsto 1] + [l_2 \mapsto \Pr(l_2)] ; [l'_1 \mapsto 1] = [l_1 l'_1 \mapsto \Pr(l_1), l_2 l'_1 \mapsto \Pr(l_2)] \\
\sigma_{12} &= [l_1 \mapsto \Pr(l_1)] ; [l'_3 \mapsto 1] + [l_2 \mapsto \Pr(l_2)] ; [l'_1 \mapsto 1] = [l_1 l'_3 \mapsto \Pr(l_1), l_2 l'_1 \mapsto \Pr(l_2)] \\
\sigma_{21} &= [l_1 \mapsto \Pr(l_1)] ; [l'_2 \mapsto 1] + [l_2 \mapsto \Pr(l_2)] ; [l'_1 \mapsto 1] = [l_1 l'_2 \mapsto \Pr(l_1), l_2 l'_1 \mapsto \Pr(l_2)] \\
\sigma_{22} &= [l_1 \mapsto \Pr(l_1)] ; [l'_2 \mapsto 1] + [l_2 \mapsto \Pr(l_2)] ; [l'_1 \mapsto 1] = [l_1 l'_2 \mapsto \Pr(l_1), l_2 l'_1 \mapsto \Pr(l_2)]
\end{aligned}$$

and so, $\mathcal{L}(P)$ is

	$l_1 l'_1$	$l_1 l'_2$	$l_1 l'_3$	$l_2 l'_1$
$h_1 h'_1$	$\Pr(l_1)$	0	0	$\Pr(l_2)$
$h_1 h'_2$	0	0	$\Pr(l_1)$	$\Pr(l_2)$
$h_2 h'_1$	0	$\Pr(l_1)$	0	$\Pr(l_2)$
$h_2 h'_2$	0	$\Pr(l_1)$	0	$\Pr(l_2)$

Thus, $\Pr(h_1 h'_1 \mid l_1 l'_1) = 1$, since $\Pr(l'_1 \mid l_1) = \Pr(h_1 h'_1)$: indeed, by (22), 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 $\Pr(h_1 h'_2 \mid l_1 l'_3)$ is similar. Let us consider now $\Pr(h_2 h'_k \mid l_1 l'_2)$, 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

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

is 2, no matter 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 about 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 moving from P to Q, and comparing the two implementations of the auction system from the security point of view. With (23), we have, e.g. that

$$\sigma_{11} = [l_1 \mapsto \Pr(l_1)]; \left[l'_1 \mapsto \frac{1}{3}, l'_2 \mapsto \frac{1}{3}, l'_3 \mapsto \frac{1}{3}\right] + [l_2 \mapsto \Pr(l_2)]; [l'_1 \mapsto 1].$$

Similarly, we can calculate σ_{12}, σ_{21} and σ_{22}. Thus, $\mathcal{L}(Q)$ 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 generally has more possible high traces associated with each low trace, and the latter are now 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. Chatzikokolakis *et al.* (2008a)) of $\mathcal{L}(Q)$ is less than the capacity of $\mathcal{L}(P)$.

6.2. *Modular exponentiation algorithm*

In the 90's, Kocher showed that RSA and other public-key cryptosystems are subject to side-channel attacks exploiting information leaked by implementations of the modular exponentiation algorithm, such as execution time (Kocher 1996) and/or power consumption (Kocher *et al.* 1999). The basic idea is that, by sampling these quantities through repeated executions of the algorithm, the attacker may be able to correlate the observed information with the secret key, that is, the bits of the exponent. The modular exponentiation algorithm we consider is shown in Algorithm 1. There, a is the basis, $d = (d_{n-1}, \ldots, d_0)$ represents an n-bit secret exponent and m is the modulus; the algorithm returns $a^d \bmod m$.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Algorithm 1 Exponentiation algorithm

```
1: input(a, d, m)
2: A := 1
3: for i = n − 1 downto 0 do
4:     A := A^2 mod m
5:     if d_i = 1 then A := a × A mod m
6: end for
7: return A;
```

We next give an abstract description of this algorithm in our language. We concentrate on leaks due to timing: from the point of view of the attacker, the only observable information is represented by execution time. This is practically the case when the algorithm is implemented with the *blinding* technique, which hides from the attacker the identity of the plaintext a that is actually processed.

The elapse of a discrete time interval is modelled as the occurrence of a low action t. We assume that the only operations in the algorithm taking non-zero time are the squaring, $A^2 \bmod m$, and product, $a \times A \bmod m$, operations. The execution times of these operations correspond to independent random variables taking values in $\mathbb{N}$, according to given probability distributions $p = (p_0, p_1, \ldots)$ and $q = (q_0, q_1, \ldots)$, respectively. The actual length of the exponent, n, is unknown to the attacker.

We assume a high alphabet $H = \{h_0, h_1\}$ and a low alphabet $L = \{t\}$. Hence, a high trace π just represents a block of bits (the secret exponent), while a low trace λ represents an execution time. We set the underlying semiring to $\mathbb{WL}$ with $L = \{t\}$, so we are actually using streams of real numbers that model probability distributions over $\mathbb{N}$. The algorithm under consideration is modelled by the term P below, where $o_p = [t^i \mapsto p_i]_{i \geqslant 0}$ and $o_q = [t^i \mapsto q_i]_{i \geqslant 0}$:

$$P \;=\; \big(\, o_p \,;\, (h_0 + o_q ; h_1)\,\big)^* .$$

The semantics of P, $\mathcal{L}(P)$, can be described by a matrix with rows indexed by high traces $\pi \in \{h_0, h_1\}^*$, representing the choice of the secret exponent, and columns indexed by low traces λ of the form t^k, for $k \geqslant 0$, representing execution times.

To examine a concrete case, suppose we want to display the sub-matrix corresponding to the length of the exponent $n = 2$. To simplify notation, and using Theorem 3, we can write the FPS $\mathcal{L}(P) = \sigma$ as

$$\sigma = (o_p ; (h_0 + o_q ; h_1))^* .$$

We want to compute $\sigma(w)$, for $w \in \{h_0h_0, h_0h_1, h_1h_0, h_1h_1\}$. We first have to compute σ_{h_0} and σ_{h_1}:

$$\begin{aligned}
\sigma_{h_0} \;&=\; (o_p ; (h_0 + o_q ; h_1))_{h_0} \,;\, \sigma \\
&=\; \big((o_p)_{h_0} ; (h_0 + o_q ; h_1) + o_p(\epsilon) \times ((h_0)_{h_0} + (o_q)_{h_0} ; h_1 + o_q(\epsilon) \times (h_1)_{h_0})\big) \,;\, \sigma \\
&=\; \big(0_{\mathbb{F}} ; (h_0 + h_1 ; o_q) + o_p \times (1_{\mathbb{F}} + 0_{\mathbb{F}} ; h_1 + o_q \times 0_{\mathbb{F}})\big) \,;\, \sigma \\
&=\; o_p ; \sigma .
\end{aligned}$$

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

	t^0	t^1	t^2	t^3	t^4	t^5	t^6	t^7	t^8	t^9	t^{10}	t^{11}	t^{12}	t^{13}	t^{14}	t^{15}	t^{16}
h_0h_0	0	0	$\frac{1}{2^6}$	$\frac{6}{2^6}$	$\frac{15}{2^6}$	$\frac{20}{2^6}$	$\frac{15}{2^6}$	$\frac{6}{2^6}$	$\frac{1}{2^6}$	0	0	0	0	0	0	0	0
h_0h_1	0	0	0	$\frac{1}{2^9}$	$\frac{9}{2^9}$	$\frac{36}{2^9}$	$\frac{84}{2^9}$	$\frac{126}{2^9}$	$\frac{126}{2^9}$	$\frac{84}{2^9}$	$\frac{36}{2^9}$	$\frac{9}{2^9}$	$\frac{1}{2^9}$	0	0	0	0
h_1h_0	0	0	0	$\frac{1}{2^9}$	$\frac{9}{2^9}$	$\frac{36}{2^9}$	$\frac{84}{2^9}$	$\frac{126}{2^9}$	$\frac{126}{2^9}$	$\frac{84}{2^9}$	$\frac{36}{2^9}$	$\frac{9}{2^9}$	$\frac{1}{2^9}$	0	0	0	0
h_1h_1	0	0	0	0	$\frac{1}{2^{12}}$	$\frac{12}{2^{12}}$	$\frac{66}{2^{12}}$	$\frac{220}{2^{12}}$	$\frac{495}{2^{12}}$	$\frac{792}{2^{12}}$	$\frac{924}{2^{12}}$	$\frac{792}{2^{12}}$	$\frac{495}{2^{12}}$	$\frac{220}{2^{12}}$	$\frac{66}{2^{12}}$	$\frac{12}{2^{12}}$	$\frac{1}{2^{12}}$

Fig. 1. Matrix for the algorithm for exponent length $n = 2$ and $p = q = (0, \frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}, 0, 0, \ldots)$.

$$\begin{aligned}
\sigma_{h_1} &= (o_p; (h_0 + o_q; h_1))_{h_1} \; ; \sigma \\
&= \big((o_p)_{h_1}; (h_0 + o_q; h_1) + o_p(\epsilon) \times ((h_0)_{h_1} + (o_q)_{h_1}; h_1 + o_q(\epsilon) \times (h_1)_{h_1})\big) \; ; \sigma \\
&= \big(0_{\mathbb{F}}; (h_0 + h_1; o_q) + o_p \times (0_{\mathbb{F}} + 0_{\mathbb{F}}; h_1 + o_q \times 1_{\mathbb{F}})\big) \; ; \sigma \\
&= (o_p \times o_q); \sigma \, .
\end{aligned}$$

As usual, in the last step of both these calculations we have used some simplifying laws. Now, by similar calculations:

$$\begin{aligned}
\sigma_{h_0h_0} &= (o_p; \sigma)_{h_0} = (o_p)_{h_0}; \sigma + o_p(\epsilon) \times \sigma_{h_0} = o_p \times (o_p; \sigma) = o_p^2; \sigma \\
\sigma_{h_0h_1} &= (o_p; \sigma)_{h_1} = (o_p)_{h_1}; \sigma + o_p(\epsilon) \times \sigma_{h_1} = o_p \times ((o_p \times o_q); \sigma) = (o_p^2 \times o_q); \sigma \\
\sigma_{h_1h_0} &= ((o_p \times o_q); \sigma)_{h_0} = (o_p \times o_q)_{h_0}; \sigma + (o_p \times o_q)(\epsilon) \times \sigma_{h_0} = (o_p^2 \times o_q); \sigma \\
\sigma_{h_1h_1} &= ((o_p \times o_q); \sigma)_{h_1} = (o_p \times o_q)_{h_1}; \sigma + (o_p \times o_q)(\epsilon) \times \sigma_{h_1} = (o_p^2 \times o_q^2); \sigma \, .
\end{aligned}$$

If we take $p = q$, we have that

$$\sigma(h_0h_0) = \mathrm{w}(\sigma_{h_0h_0}) = o_p^2 \qquad \sigma(h_0h_1) = \sigma(h_1h_0) = \mathrm{w}(\sigma_{h_0h_1}) = o_p^3 \qquad \sigma(h_1h_1) = \mathrm{w}(\sigma_{h_1h_1}) = o_p^4.$$

Note that since $\mathbb{S}$ is the semiring of positive real streams (or, equivalently, the semiring $\mathbb{WL}$ with $|L| = 1$), according to (2) the product $\times$ is just convolution between streams. This operation – and all other operations between elements of $\mathbb{WL}$ – can in turn be computed by the same method outlined above, streams being just a special case of FPS's.

If we assume that $p = q$ is distributed over $0 \ldots N$, for some $N \geqslant 1$, then, for any fixed length n of the exponent, the sub-matrix of σ given by all the rows π of length n is finite, in the sense that the columns λ of this sub-matrix exceeding length $2 \times n \times N$ are everywhere 0.

Figure 1 displays the whole (non-zero) matrix when $p = q$ are binomially distributed over $0 \ldots 4$.[†]

6.3. *Imperative computations*

This section provides a different way of writing examples; indeed, instead of adopting a process algebraic flavour (like, e.g. in Sections 6.1 and 6.2), we adopt here a more

[†] That is, $p_i = q_i = \binom{3}{i-1} \times 2^{-3}$, for $i = 0, 1, 2, \ldots$. Note that $p_i = q_i = 0$ for $i = 0$ and $i > 4$. Here, the binomial distribution is meant as a quantized version of the Gaussian distributions often found in a continuous setting, see Kocher (1996).

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

imperative flavour, by exploiting the semiring of stores $\mathbb{M}$ described in Section 2. We let $\mu, \mu', \ldots$ 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 that typically occur 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 defined by

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

We just have to prove that such a function is a semiring (endo)morphism. All properties are trivial, except for preservation of product. We have to show that $f_{y:=x+1}(\mu \star \mu') = f_{y:=x+1}(\mu) \star f_{y:=x+1}(\mu')$. Let $m \in f_{y:=x+1}(\mu \star \mu')$, i.e. $m = m_1 \star m_2 \star [y = (m_1 \star m_2)(x) + 1]$, if $(m_1 \star m_2)(x)$ is defined and $m = m_1 \star m_2$, otherwise; in both cases, $m_1 \in \mu$ and $m_2 \in \mu'$. If $(m_1 \star m_2)(x)$ is undefined, then both $m_1(x)$ and $m_2(x)$ are undefined; thus, $m_1 \in f_{y:=x+1}(\mu)$ and $m_2 \in f_{y:=x+1}(\mu')$, that implies $m \in f_{y:=x+1}(\mu) \star f_{y:=x+1}(\mu')$. If $(m_1 \star m_2)(x)$ is defined, then it can be that

1. either $m_2(x)$ is defined (and in this case $(m_1 \star m_2)(x) = m_2(x)$)
2. or that $m_2(x)$ is undefined (and in this case $m_1(x)$ is defined and $(m_1 \star m_2)(x) = m_1(x)$).

In case 1, let $\bar{m}_1$ be $m_1 \star [y = m_1(x) + 1]$, if $m_1(x)$ is defined, and be m_1, otherwise; then, $m = \bar{m}_1 \star (m_2 \star [y = m_2(x) + 1]) \in f_{y:=x+1}(\mu) \star f_{y:=x+1}(\mu')$. In case 2, $m = (m_1 \star [y = m_1(x) + 1]) \star m_2 \in f_{y:=x+1}(\mu) \star f_{y:=x+1}(\mu')$. The converse inclusion can be proved in a similar way.

Similarly to variable assignments, also conditionals can be modelled through filters. For example, the program fragment P; if $(x \neq y)$ then $y := y + 1$ else $z := 1$ corresponds to the term

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

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

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

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

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

$f_L(\mu) \triangleq \{m|_{V_L} \ : \ m \in \mu\}$, where $m|_{V_L}$ denotes the restriction of m to V_L. 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 (e.g. $h = 0$ is modelled by h_0): 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

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

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 (digit-by-digit) and then stores it into a high variable h. The attacker chooses a guess for this PIN (again, digit-by-digit) and stores it into a low variable l. This behaviour is modelled by

$$\begin{aligned} Choose_{secr} &\triangleq \textstyle\sum_{i=0}^{9} h_i; \sum_{j=0}^{9} h_j; \sum_{k=0}^{9} h_k; \sum_{q=0}^{9} h_q; [h = ijkq] \\ Choose_{guess} &\triangleq \textstyle\sum_{i=0}^{9} \sum_{j=0}^{9} \sum_{k=0}^{9} \sum_{q=0}^{9} [l = ijkq] \\ Choose &\triangleq Choose_{secr}; Choose_{guess}. \end{aligned}$$

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:

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

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 $h_i h_j h_k h_q$ to the set of stores $\mu_{ijkq} = \{[l = ijkq, r = ok]\} \cup \{[l = g, r = no] \ : \ g \neq ijkq\}$; therefore, for $s \neq s'$, we have $\mu_s \neq \mu_{s'}$. We could make the behaviour of the PIN checker more refined, by e.g. combining the two semirings considered in Section 2 and associating 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 (cf. Hillston (1996) and references therein). 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 (1996) and other authors doing stochastic process algebra. In fact, it is technically easy to extend the language presented in Section 3 with operators that introduce some form of parallelism. The corresponding operational rules mimic those found in process calculi, e.g. CSP (Boreale and Gadducci 2006). As a further simplification, in the following we shall confine ourselves to a pure interleaving operator, $||$. Thus, we extend the process syntax

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

of (3) and process weight of (9) as:

$$P ::= \ldots \mid P||Q \tag{24}$$

$$\mathrm{w}(P||Q) = \mathrm{w}(P) \times \mathrm{w}(Q)\,. \tag{25}$$

Moreover, we 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, if $\Delta = \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(P)$, then:

$$\Delta||Q \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(P||Q) \tag{26}$$

$$Q||\Delta \triangleq \sum_{P \in \mathcal{P}} \Delta(P) \times \delta(Q||P)\,. \tag{27}$$

In the final semantics, this corresponds to the *shuffle* operator on FPS defined by the following BDE, where $h \in H$:

$$(\sigma||\tau)(\epsilon) = \sigma(\epsilon) \times_{\mathbb{S}} \tau(\epsilon) \qquad (\sigma||\tau)_h = (\sigma_h||\tau) + (\sigma||\tau_h)\,. \tag{28}$$

As in Theorem 3, there exists a unique operator in $\mathbb{F}$ that satisfies this BDE; moreover, as in Corollary 2, such an operator satisfies $\mathcal{L}(P||Q) = \mathcal{L}(P)||\mathcal{L}(Q)$.

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 $\pi \in H^*$ 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 $|\pi|$, viz. $\{l, l'\}^{|\pi|}$.

8. Implementation

We discuss here some aspects concerning the implementation of our approach. We focus on the computation of the language function $\mathcal{L}(P) : H^* \to \mathbb{S}$, for any given process P: this is the central object of interest to a designer. One way of implementing this function is to just translate into code the operational semantics of Table 1. Here, we want to discuss an alternative view that leads to an implementation in terms of lazy data structures and operators, as found for instance in the language Haskell (see e.g. Douglas Mcilroy (1999)). This implementation, we think, is interesting because it is direct, elegant and memory efficient.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

It is then convenient to work directly in terms of FPS's and operators over them. In particular, if $\sigma = \mathcal{L}(P)$, then $\mathcal{L}(P)(w)$ is just $\sigma(w) = \sigma_w(\epsilon)$, for any $w \in H^*$. Recall that, in order to compute $\sigma(w)$ and assuming that σ is described by a regular expression, we can:

1. compute σ_w, the w-derivative of σ, by repeatedly applying in a syntax-driven way the equations in the second column of the BDE's table from left to right (Table 2). This rewrite stops as soon as any h-derivative operator σ_h in the expression is pushed underneath a sequential composition operator (; or equivalently $\times$);
2. compute $\sigma_w(\epsilon)$, again in a syntax-driven way, by the equations in the first column of the BDE's in the same table.

As far as an actual implementation of this method is concerned, it is useful to view FPS's as infinite trees, with nodes labelled over $\mathbb{S}$ and arcs labelled over the alphabet H: the FPS σ is represented by the tree having the root labelled by $\sigma(\epsilon)$ and an h-labelled arc leading to the derivative σ_h for each $h \in H$. These kind of trees can be implemented in any language supporting lazy evaluation. In Haskell, the corresponding data-type might be defined as follows. Here we assume $H = \{h_0, h_1\}$, hence binary trees, for simplicity:

```
data FPS a = Const a (FPS a) (FPS a)
```

where `a` is a parameter representing the type of the semiring $\mathbb{S}$, while `Const` represents the constant coefficient of the FPS, that is $\sigma(\epsilon)$. On these trees, it is trivial to define the following operations:

— `eps f`, the operation that extracts the constant coefficient of `f`;
— `h0 f` and `h1 f`, the operations that yield the h_0- and h_1-derivative (that is, the left and right sub-tree) of `f`, respectively;
— and `[k]`, the operation that yields the series that has `k` as a constant coefficient and is $0_{\mathbb{S}}$ elsewhere.

The definitions of the operators in our calculus are then a direct translation of the corresponding BDE's: it is sufficient to describe the constant term ($\sigma(\epsilon)$) and each of the derivatives (σ_{h_0} and σ_{h_1}). Specifically, the operators of sum (+), product (.) and iteration (it) are defined thus:

```
f1+f2 = Const ((eps f1) S.+ (eps f2)) ((h0 f1)+(h0 f2)) ((h1 f1)+(h1 f2))

f1.f2 = Const ((eps f1) S.* (eps f2))
              ((h0 f1).f2 + [eps f1] . (h0 f2))
              ((h1 f1).f2 + [eps f1] . (h1 f2))

it f = (Const S.u) ((h0 f).(it f)) ((h1 f).(it f)).
```

Here, `S.+`, `S.*` and `S.u` represent the sum, product and unit element, respectively, of the underling semiring $\mathbb{S}$, as defined in a module `S`.

The lazy evaluation mechanism automatically takes care of performing the kind of step wise expansion we have manually computed in the examples of Section 6. For instance, if `t` is a term corresponding to the modular exponentiation process P seen in Section 6.2, $\mathcal{L}(P)(h_0h_1)$ could be extracted by

```
eps (h1 (h0 t)).
```

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Note that, in case $\mathbb{S}$ is the semiring of streams, the above code would yield a stream, which is still an infinite object. We may rather want to take only the first, say, 17 elements of it, which is exactly the second row of the matrix in Figure 1. This can be done using the Haskell function `take`, thus:

```
take 17  (eps (h1 (h0 t))).
```

The source code of a complete Haskell implementation of our calculus along the lines outlined above can be found at http://rap.dsi.unifi.it~boreale/Haskell.rar[†]. Most of the examples described in the paper have been tested using this implementation and the corresponding code can be found there.

9. Concluding remarks

In the last ten years, there has been steady activity in developing concepts, definitions and analyses in the area of measuring information flows for different models. 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 led to some notable progress for simple imperative languages (Clark *et al.* 2007; Heusser and Malacaria 2010; Köpf and Rybalchenko 2010; Mu and Clark 2009a,b). By contrast, progress for process algebras has been notably slower. One problem has been establishing appropriate concepts. The work of Lowe (2002) provided a starting point, developed in quite diverse directions by many authors. Mostly related to our work are those papers developing quantitative information flow (QIF) in various interactive scenarios – see e.g. Backes (2005), Boreale (2009), Chatzikokolakis *et al.* (2008a), Chatzikokolakis *et al.* (2008b), Mu (2009) and references therein. In comparison with the latter body of work, the present paper makes a conceptual leap, rather than simply a 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 stress on coalgebras and languages as tools for security analysis is novel as well. Also related to our research is the process-algebraic approach to non-interference initiated by Focardi and Gorrieri (1995) and developed in a quantitative direction notably by Aldini *et al.* in a series of papers – see e.g. Aldini *et al.* (2002, 2004); Aldini and Di Pierro (2004) and references therein. An important difference between our work and this line of research is that we depart from non-interference to explicitly endorse the typical QIF view: systems are seen as probabilistic (or possibilistic) *channels*, representable as matrices, for which one aims to analyse the amount and type of information that can be passed from the secrets to the observables. Hence, our use of information-theoretic conceptual tools and terminology.

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 Rutten (2003). In a recent paper (Bonchi

[†] Due to certain idiosyncrasies of Haskell syntax, the names given to some operators there are different from those used here.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

et al. 2009), they present a systematic way to generate languages of (generalized) 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 different objects, related to the definition of 'good parallel operator'. Given their branching-time nature, these objects lack an interpretation in terms of traces, languages and security analysis as proposed in the present paper. Second, they focus on axiomatizations rather than on compositional semantics in terms of rational operators and BDE's, as we do here.

Our work is closely related to Wittbold and Johnson (1990) in which the authors study different security properties for nondeterministic state machines. They distinguish two levels of nondeterminism: the *possibilistic* and the *probabilistic* one. The possibilistic level is more abstract, in the sense that it only collects the 'possible' outcomes of a system. For this level, several yes-or-no notions of security are examined; in particular, the focus is put on the notion of *nondeducibility on strategies*, whereby a system is secure if and only if any visible view of the system is consistent with any hidden strategy. There are echos of these ideas in our approach, as a given strategy can be associated with a prefix closed set of secret traces. Since we consider observations at termination, each possible strategy is associated with a single secret trace π and the visible view is our observable sequence λ. A probabilistic variant of nondeducibility on strategies can then be obtained in our framework by requiring that matrix $\mathcal{L}(P)$ has constant columns (or, equivalently, all rows are the same). When considering probabilistic nondeterminism, (Wittbold and Johnson 1990) uses a representation very similar to $\mathcal{L}(P)$. This is used to model a system as a channel (in the information theoretic sense), whose zero-capacity entails nondeducibility on strategies. We note that their paper contains a counterexample demonstrating that the property is insufficient to guarantee the absence of covert channels in the presence of (information theoretic) noise. Our work provides compositional and syntax-directed ways for automatically calculating the channel matrix, whereas in Wittbold and Johnson (1990) this object is extracted from the definition of the system specification directly by mathematical calculations.

In the field of semantics for probabilistic transition systems, our work appears to be related to Desharnais *et al.* (2002). There the authors define a notion of equivalence for probabilistic processes with the aim of proving it a congruence with respect to the usual process algebra operators. As an application, the authors suggest the possibility of calculating a matrix that describes the correlations between observable traces (our λ) and secret inputs (our π); this is done is the usual way, i.e. by considering a process as a channel (in the sense of information theory), by calculating its matrix and its capacity. Again, this is similar to our approach, but no compositional and syntax-driven construction of the matrix is provided.

In Andrés *et al.* (2010), the authors face a problem similar to ours in a different way. There, they consider a system as a fully probabilistic automaton, consider it as a channel (in the information-theoretic sense) and try to calculate its associated matrix. They give two different approaches, one based on solving a system of linear equations, the other based on an incremental procedure. In both cases, they start with a simple model, where

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

all secret choices are known and taken at the beginning; then they refine the model to let the secrets interleave freely with the observable events. Their model is different, since we aimed at modelling a more general notion of weight than the one put forward by probabilities. Moreover, our approach was intended to be syntax-driven and so we started from a language of process expressions, whereas in Andrés *et al.* (2010) the system is given as an automaton and all calculations are done by exploiting reachability or path analysis. In the same spirit as Andrés *et al.* (2010), it is worth mentioning also Chatzikokolakis *et al.* (2008a), where model checking techniques are used to calculate the same notion of channel capacity for the simplest model of probabilistic automata (namely, the one where every choice is taken at the beginning).

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

We thank the anonymous reviewers for their suggestions and comments that helped us in improving the paper. The research was partly supported by the EU within the FET-GC2 initiative, project ASCENS as well as 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 partly supported by the UK EPSRC project EP/C545605/1, Quantified Information Flow.

Appendix A. Technical Proofs

Proof of Proposition 1

The first three items directly come from the definitions.

4.
$$\begin{aligned}(o \times \Delta)\langle f\rangle &\triangleq \textstyle\sum_{R\in\mathcal{P}}((o \times \Delta)\langle f\rangle)(R) \times \delta(R)\\ &\triangleq \textstyle\sum_{R'\in\mathcal{P}} f((o \times \Delta)(R')) \times \delta(R'\langle f\rangle)\\ &= \textstyle\sum_{R'\in\mathcal{P}} f(o \times \Delta(R')) \times \delta(R'\langle f\rangle)\\ &= \textstyle\sum_{R'\in\mathcal{P}} f(o) \times f(\Delta(R')) \times \delta(R'\langle f\rangle)\\ &= f(o) \times \textstyle\sum_{R'\in\mathcal{P}} f(\Delta(R')) \times \delta(R'\langle f\rangle)\\ &\triangleq f(o) \times \textstyle\sum_{R\in\mathcal{P}}(\Delta\langle f\rangle)(R) \times \delta(R)\\ &\triangleq f(o) \times \Delta\langle f\rangle\end{aligned}$$

where the first equality directly comes from (5), the second equality is ensured by the fact that f is a semiring morphism and the third one by associativity and distributivity of the semiring.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

5. $$\begin{aligned}(\Delta_1+\Delta_2)\langle f\rangle &\triangleq \textstyle\sum_{R\in\mathcal{P}}((\Delta_1+\Delta_2)\langle f\rangle)(R)\times\delta(R)\\ &\triangleq \textstyle\sum_{R'\in\mathcal{P}} f((\Delta_1+\Delta_2)(R'))\times\delta(R'\langle f\rangle)\\ &= \textstyle\sum_{R'\in\mathcal{P}} f(\Delta_1(R')+\Delta_2(R'))\times\delta(R'\langle f\rangle)\\ &= \textstyle\sum_{R'\in\mathcal{P}}(f(\Delta_1(R'))+f(\Delta_2(R')))\times\delta(R'\langle f\rangle)\\ &= \textstyle\sum_{R'\in\mathcal{P}} f(\Delta_1(R'))\times\delta(R'\langle f\rangle)\ +\ \sum_{R'\in\mathcal{P}} f(\Delta_2(R'))\times\delta(R'\langle f\rangle)\\ &= \textstyle\sum_{R\in\mathcal{P}}(\Delta_1\langle f\rangle(R))\times\delta(R)\ +\ \sum_{R\in\mathcal{P}}(\Delta_2\langle f\rangle(R))\times\delta(R)\\ &\triangleq \Delta_1\langle f\rangle+\Delta_2\langle f\rangle\end{aligned}$$

where the first equality directly comes from (4), the second equality is ensured by the fact that f is a semiring morphism and the third one by distributivity and associativity of the semiring.

Proof of Lemma 1

1. We prove that $\mathcal{R}\triangleq\{(\Delta+\Delta_1,\Delta+\Delta_2)\ :\ \Delta_1\sim\Delta_2\}$ is a bisimulation.

— The output is the same:

$$\begin{aligned}\mathrm{w}(\Delta+\Delta_1) &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta+\Delta_1)(R)\times\mathrm{w}(R)\\ &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta(R)+\Delta_1(R))\times\mathrm{w}(R)\\ &= \textstyle\sum_{R\in\mathcal{P}}\Delta(R)\times\mathrm{w}(R)+\sum_{R\in\mathcal{P}}\Delta_1(R)\times\mathrm{w}(R)\\ &\triangleq \mathrm{w}(\Delta)+\mathrm{w}(\Delta_1)\\ &= \mathrm{w}(\Delta)+\mathrm{w}(\Delta_2)\\ &= \mathrm{w}(\Delta+\Delta_2)\end{aligned}$$

where the first equality holds because of distributivity in the semiring and the second equality holds because $\mathrm{w}(\Delta_1)=\mathrm{w}(\Delta_2)$, being $\Delta_1\sim\Delta_2$.

— The derivatives are still bisimilar:

$$\begin{aligned}(\Delta+\Delta_1)_h &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta+\Delta_1)(R)\times\Delta_{R,h}\\ &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta(R)+\Delta_1(R))\times\Delta_{R,h}\\ &= \textstyle\sum_{R\in\mathcal{P}}((\Delta(R)\times\Delta_{R,h})+(\Delta_1(R)\times\Delta_{R,h}))\\ &= (\textstyle\sum_{R\in\mathcal{P}}\Delta(R)\times\Delta_{R,h})+(\sum_{R\in\mathcal{P}}\Delta_1(R)\times\Delta_{R,h})\\ &\triangleq \Delta_h+(\Delta_1)_h\end{aligned}$$

where the first equality holds because of Proposition 1(1). Similarly, $(\Delta+\Delta_2)_h=\Delta_h+(\Delta_2)_h$. Since $(\Delta_1)_h\sim(\Delta_2)_h$, we conclude that $((\Delta+\Delta_1)_h,(\Delta+\Delta_2)_h)\in\mathcal{R}$.

2. We prove that $\mathcal{R}\triangleq\{(o\times\Delta_1,o\times\Delta_2)\ :\ \Delta_1\sim\Delta_2\}$ is a bisimulation.

— The output is the same:

$$\begin{aligned}\mathrm{w}(o\times\Delta_1) &\triangleq \textstyle\sum_{R\in\mathcal{P}}(o\times\Delta_1)(R)\times\mathrm{w}(R)\\ &\triangleq \textstyle\sum_{R\in\mathcal{P}}(o\times\Delta_1(R))\times\mathrm{w}(R)\\ &= o\times\textstyle\sum_{R\in\mathcal{P}}\Delta_1(R)\times\mathrm{w}(R)\\ &\triangleq o\times\mathrm{w}(\Delta_1)\\ &= o\times\mathrm{w}(\Delta_2)\\ &= \mathrm{w}(o\times\Delta_2)\end{aligned}$$

since $\mathrm{w}(\Delta_1)=\mathrm{w}(\Delta_2)$ being $\Delta_1\sim\Delta_2$.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

— The derivatives are still bisimilar:

$$\begin{aligned}(o \times \Delta_1)_h &\triangleq \textstyle\sum_{R\in\mathcal{P}}(o \times \Delta_1)(R) \times \Delta_{R,h}\\ &\triangleq \textstyle\sum_{R\in\mathcal{P}}(o \times \Delta_1(R)) \times \Delta_{R,h}\\ &= \textstyle\sum_{R\in\mathcal{P}} o \times (\Delta_1(R) \times \Delta_{R,h})\\ &= o \times \textstyle\sum_{R\in\mathcal{P}} \Delta_1(R) \times \Delta_{R,h}\\ &\triangleq o \times (\Delta_1)_h\end{aligned}$$

where the two equalities are due to Proposition 1(3,2). Similarly, $(o\times\Delta_2)_h = o\times(\Delta_2)_h$. Since $(\Delta_1)_h \sim (\Delta_2)_h$, we conclude that $((o \times \Delta_1)_h, (o \times \Delta_2)_h) \in \mathcal{R}$.

3. We prove that

$$\mathcal{R} \triangleq \{((\Delta_1; P) + \Delta' \,,\, (\Delta_2; Q) + \Delta'') \;:\; \Delta_1 \sim \Delta_2 \text{ and } P \sim Q \text{ and } \Delta' \sim \Delta''\}$$

is a bisimulation; this trivially implies the result by taking $\Delta' = \Delta'' = 0_{\mathcal{M}}$.

— The output is the same:

$$\begin{aligned}\mathrm{w}((\Delta_1; P) + \Delta') &\triangleq \textstyle\sum_{R\in\mathcal{P}}((\Delta_1; P) + \Delta')(R) \times \mathrm{w}(R)\\ &\triangleq \textstyle\sum_{R\in\mathcal{P}}((\Delta_1; P)(R) + \Delta'(R)) \times \mathrm{w}(R)\\ &= \textstyle\sum_{R\in\mathcal{P}}(\Delta_1; P)(R) \times \mathrm{w}(R) + \sum_{R\in\mathcal{P}} \Delta'(R) \times \mathrm{w}(R)\\ &\triangleq \textstyle\sum_{R=R';P} \Delta_1(R') \times \mathrm{w}(R) \quad + \quad \mathrm{w}(\Delta')\\ &= \textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times \mathrm{w}(R'; P) \quad + \quad \mathrm{w}(\Delta')\\ &\triangleq \textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times \mathrm{w}(R') \times \mathrm{w}(P) \quad + \quad \mathrm{w}(\Delta')\\ &= \mathrm{w}(P) \times \mathrm{w}(\Delta_1) \quad + \quad \mathrm{w}(\Delta').\end{aligned}$$

Similarly, $\mathrm{w}((\Delta_2; Q) + \Delta'') = \mathrm{w}(Q) \times \mathrm{w}(\Delta_2) \;+\; \mathrm{w}(\Delta'')$. Now, $\mathrm{w}(P) = \mathrm{w}(Q)$, because $P \sim Q$, $\mathrm{w}(\Delta_1) = \mathrm{w}(\Delta_2)$, because $\Delta_1 \sim \Delta_2$ and $\mathrm{w}(\Delta') = \mathrm{w}(\Delta'')$, because $\Delta' \sim \Delta''$; this suffices to conclude that $\mathrm{w}((\Delta_1; P) + \Delta') = \mathrm{w}((\Delta_2; Q) + \Delta'')$.

— The derivatives are still bisimilar:

$$\begin{aligned}&((\Delta_1; P) + \Delta')_h\\ &\quad\triangleq \textstyle\sum_{R\in\mathcal{P}}((\Delta_1; P) + \Delta')(R) \times \Delta_{R,h}\\ &\quad\triangleq \textstyle\sum_{R\in\mathcal{P}}((\Delta_1; P)(R) + \Delta'(R)) \times \Delta_{R,h}\\ &\quad= \textstyle\sum_{R\in\mathcal{P}}((\Delta_1; P)(R) \times \Delta_{R,h}) + \sum_{R\in\mathcal{P}}(\Delta'(R) \times \Delta_{R,h})\\ &\quad\triangleq \textstyle\sum_{R'\in\mathcal{P}}(\Delta_1(R') \times \Delta_{R';P,h}) + (\Delta')_h\\ &\quad= \textstyle\sum_{R'\in\mathcal{P}}(\Delta_1(R') \times (\Delta_{R',h}; P + \mathrm{w}(R') \times \Delta_{P,h}) + (\Delta')_h\\ &\quad= (\textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times (\Delta_{R',h}; P))\\ &\qquad + (\textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times (\mathrm{w}(R') \times \Delta_{P,h})) + (\Delta')_h\\ &\quad= (\textstyle\sum_{R'\in\mathcal{P}}(\Delta_1(R') \times (\Delta_{R',h})); P)\\ &\qquad + (\textstyle\sum_{R'\in\mathcal{P}}(\Delta_1(R') \times \mathrm{w}(R')) \times \Delta_{P,h}) + (\Delta')_h\\ &\quad= (\textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times (\Delta_{R',h})); P\\ &\qquad + (\textstyle\sum_{R'\in\mathcal{P}}(\Delta_1(R') \times \mathrm{w}(R')) \times \Delta_{P,h}) + (\Delta')_h\\ &\quad\triangleq ((\Delta_1)_h; P) + (\textstyle\sum_{R'\in\mathcal{P}}(\Delta_1(R') \times \mathrm{w}(R')) \times \Delta_{P,h}) + \Delta'_h\\ &\quad= ((\Delta_1)_h; P) + (\mathrm{w}(\Delta_1) \times \Delta_{P,h}) + \Delta'_h\end{aligned}$$

where the first equality holds because of Proposition 1(1), the second one derives from the operational semantics of sequential composition, the third one by Proposition 1(2), the fourth one by Proposition 1(4,3), the fifth one by

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Proposition 1(2) and the last one by Proposition 1(1). Similarly with Q in place of P, Δ'' in place of Δ' and Δ_2 in place of Δ_1. By hypothesis, $\Delta'_h \sim \Delta''_h$; moreover, the fact that $\Delta_{P,h} \sim \Delta_{Q,h}$ (that holds because $P \sim Q$), $\mathrm{w}(\Delta_1) = \mathrm{w}(\Delta_2)$ and Lemma 1(2) allow us to conclude that $\mathrm{w}(\Delta_1) \times \Delta_{P,h} \sim \mathrm{w}(\Delta_2) \times \Delta_{Q,h}$. Thus, by Lemma 1(1), $(\mathrm{w}(\Delta_1) \times \Delta_{P,h}) + \Delta'_h \sim (\mathrm{w}(\Delta_2) \times \Delta_{Q,h}) + \Delta''_h$. Thanks to Proposition 1(1), this suffices to conclude that $(((\Delta_1; P) + \Delta')_h\ , ((\Delta_2; Q) + \Delta'')_h) \in \mathcal{R}$, as required.

4. We prove that $\mathcal{R} \triangleq \{(\Delta_1\langle f\rangle, \Delta_2\langle f\rangle)\ :\ \Delta_1 \sim \Delta_2\}$ is a bisimulation.

— The output is the same:

$$\begin{aligned}
\mathrm{w}(\Delta_1\langle f\rangle) &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta_1\langle f\rangle)(R) \times \mathrm{w}(R)\\
&\triangleq \textstyle\sum_{R'\in\mathcal{P}} f(\Delta_1(R')) \times \mathrm{w}(R'\langle f\rangle)\\
&\triangleq \textstyle\sum_{R'\in\mathcal{P}} f(\Delta_1(R')) \times f(\mathrm{w}(R'))\\
&= f(\textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times \mathrm{w}(R'))\\
&\triangleq f(\mathrm{w}(\Delta_1))\\
&= f(\mathrm{w}(\Delta_2))\\
&= \mathrm{w}(\Delta_2\langle f\rangle)
\end{aligned}$$

where the fourth step has been obtained by using preservation of product and sum ensured by the fact that f is a semiring morphism.

— The derivatives are still bisimilar:

$$\begin{aligned}
(\Delta_1\langle f\rangle)_h &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\Delta_1\langle f\rangle)(R) \times \Delta_{R,h}\\
&\triangleq \textstyle\sum_{R'\in\mathcal{P}} f(\Delta_1(R')) \times \Delta_{R'\langle f\rangle,h}\\
&= \textstyle\sum_{R'\in\mathcal{P}} f(\Delta_1(R')) \times (\Delta_{R',h}\langle f\rangle)\\
&= \textstyle\sum_{R'\in\mathcal{P}}(\Delta_1(R') \times \Delta_{R',h})\langle f\rangle\\
&= (\textstyle\sum_{R'\in\mathcal{P}} \Delta_1(R') \times \Delta_{R',h})\langle f\rangle\\
&\triangleq (\Delta_1)_h\langle f\rangle
\end{aligned}$$

where the first equality comes from the definition of the operational semantics and the last two ones are due to Proposition 1(4,5). Similarly, $(\Delta_2\langle f\rangle)_h = (\Delta_2)_h\langle f\rangle$. Since $(\Delta_1)_h \sim (\Delta_2)_h$, we conclude that $((\Delta_1)_h\langle f\rangle, (\Delta_2)_h\langle f\rangle) \in \mathcal{R}$. □

Proof of Lemma 2

— The output is the same:

$$\begin{aligned}
\mathrm{w}(\delta(P) + \delta(Q)) &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\delta(P) + \delta(Q))(R) \times \mathrm{w}(R)\\
&\triangleq \textstyle\sum_{R\in\mathcal{P}}(\delta(P)(R) + \delta(Q)(R)) \times \mathrm{w}(R)\\
&= \textstyle\sum_{R\in\mathcal{P}} \delta(P)(R) \times \mathrm{w}(R) \ \ + \ \ \sum_{R\in\mathcal{P}} \delta(Q)(R) \times \mathrm{w}(R)\\
&= \delta(P)(P) \times \mathrm{w}(P) \ \ + \ \ \delta(Q)(Q) \times \mathrm{w}(Q)\\
&= \mathrm{w}(P) + \mathrm{w}(Q)\\
&\triangleq \mathrm{w}(P + Q)\\
&= \mathrm{w}(\delta(P + Q)).
\end{aligned}$$

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

— The derivative is the same:

$$\begin{aligned}(\delta(P)+\delta(Q))_h &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\delta(P)+\delta(Q))(R)\times\Delta_{R,h}\\ &\triangleq \textstyle\sum_{R\in\mathcal{P}}(\delta(P)(R)+\delta(Q)(R))\times\Delta_{R,h}\\ &\triangleq (\textstyle\sum_{R\in\mathcal{P}}\delta(P)(R)\times\Delta_{R,h})+(\textstyle\sum_{R\in\mathcal{P}}\delta(Q)(R)\times\Delta_{R,h})\\ &= \Delta_{P,h}+\Delta_{Q,h}\\ &= \Delta_{P+Q,h}\\ &= \textstyle\sum_{R\in\mathcal{P}}(\delta(P+Q))(R)\times\Delta_{R,h}\\ &\triangleq (\delta(P+Q))_h.\end{aligned}$$

Indeed, by definition of the operational semantics, $P \xrightarrow{h} \Delta_{P,h}$ and $Q \xrightarrow{h} \Delta_{Q,h}$ imply that $P+Q \xrightarrow{h} \Delta_{P,h}+\Delta_{Q,h} \triangleq \Delta_{P+Q,h}$. □

Appendix B. Overloaded Symbols

In this appendix, we provide a table with all the symbols that have been overloaded in this paper. For each of them, we provide its type (arguments/result), a use case and a reference to the appropriate definition.

	$+$	$\times$	$;$	*	$\langle f\rangle$	$\|\|$	w
$\mathbb{S}\times\mathbb{S}\to\mathbb{S}$	$o+o'$ (Definition 1)	$o\times o'$ (Definition 1)					
$\mathbb{F}\times\mathbb{F}\to\mathbb{F}$	$\sigma+\tau$ (Equation (1))	$\sigma\times\tau$ (Equation (2))	$\sigma;\tau$ (Table 2)			$\sigma\|\|\tau$ (Equation 28)	
$\mathbb{F}\to\mathbb{F}$				σ^* (Table 2)	$\sigma\langle f\rangle$ (Table 2)		
$\mathcal{P}\times\mathcal{P}\to\mathcal{P}$	$P+Q$ (Equation (3))		$P;Q$ (Equation 3)			$P\|\|Q$ (Equation 24)	
$\mathcal{P}\to\mathcal{P}$				P^* (Equation (3))	$P\langle f\rangle$ (Equation (3))		
$\mathcal{M}\times\mathcal{M}\to\mathcal{M}$	$\Delta+\Delta'$ (Equation (4))						
$\mathbb{S}\times\mathcal{M}\to\mathcal{M}$		$o\times\Delta$ (Equation (5))					
$\mathcal{M}\times\mathcal{P}\to\mathcal{M}$			$\Delta;P$ (Equation (6))			$\Delta\|\|P$ (Equation (26))	
$\mathcal{P}\times\mathcal{M}\to\mathcal{M}$			$P;\Delta$ (Equation (7))			$P\|\|\Delta$ (Equation (27))	
$\mathcal{M}\to\mathcal{M}$					$\Delta\langle f\rangle$ (Equation (8))		
$\mathcal{P}\to\mathbb{S}$							w(P) (Equation (9), (25))
$\mathcal{M}\to\mathbb{S}$							w(Δ) (Equation (10))

References

Aldini, A., Bravetti, M., Di Pierro, A., Gorrieri, R., Hankin, C. and Wiklicky, H. (2002) Two formal approaches for approximating noninterference properties. In: Proceedings of FOSAD. *Springer Lecture Notes in Computer Science* **2946** 1–43.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Aldini, A., Bravetti, M. and Gorrieri, R. (2004) A process-algebraic approach for the analysis of probabilistic noninterference. *Journal of Computer Security* **12** (2) 191–245.

Aldini, A. and Di Pierro, A. (2004) A quantitative approach to noninterference for probabilistic systems. *Electronic Notes in Theoretical Computer Science* **99** 155–182.

Andrés, M. E., Palamidessi, C., van Rossum, P. and Smith, G. (2010) Computing the leakage of information-hiding systems. In: Proceedings of TACAS. *Springer Lecture Notes in Computer Science* **6015** 373–389.

Backes, M. (2005) Quantifying probabilistic information flow in computational reactive systems. In: Proceedings of ESORICS. *Springer Lecture Notes in Computer Science* **3679** 336–354.

Bonchi, F., Bonsangue, M., Rutten, J. and Silva, A. (2009) Deriving syntax and axioms for quantitative regular behaviours. In: Proceedings of CONCUR. *Springer Lecture Notes in Computer Science* **5710** 146–162.

Boreale, M., Clark, D. and Gorla, D. (2010) A semiring-based trace semantics for processes with applications to information leakage analysis. In: Proceedings of IFIP-TCS. *IFIP AICT* **323** 340–354.

Boreale, M. and Gadducci, F. (2006) Processes as formal power series: A coinductive approach to denotational semantics. *Theoretical Computer Science* **360** (1-3) 440–458.

Boreale, M. (2009) Quantifying information leakage in process calculi. *Information and Computation* **207** (6) 699–725.

Chatzikokolakis, K., Palamidessi, C. and Panangaden, P. (2008a) Anonymity protocols as noisy channels. *Information and Computation* **206** (2-4) 378–401.

Chatzikokolakis, K., Palamidessi, C. and Panangaden, P. (2008b) On the bayes risk in information-hiding protocols. *Journal of Computer Security* **16** (5) 531–571.

Chong, S. and Myers, A. C. (2004) Security policies for downgrading. In: *Proceedings of CCS*, ACM Press 189–209.

Clark, D., Hunt, S. and Malacaria, P. (2001) Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science* **59** (3) 238–251.

Clark, D., Hunt, S. and Malacaria, P. (2007) A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security* **15** (3) 321–371.

Desharnais, J., Jagadeesan, R., Gupta, V. and Panangaden, P. (2002) The metric analogue of weak bisimulation for probabilistic processes. In: *Proceedings of LICS*, IEEE Computer Society Press 413–422.

Douglas Mcilroy, M. (1999) Power series, power serious. *Journal of Functional Programming* **9** (3) 325–337.

Focardi, R. and Gorrieri, R. (1995) A classification of security properties for process algebras. *Journal of Computer Security* **3** (1) 5–33.

Goguen, J. and Meseguer, J. (1982) Security policies and security models. In: *Proceedings of Symposium on Security and Privacy*, IEEE Computer Society Press 11–20.

Heusser, J. and Malacaria, P. (2010) Quantifying information leaks in software. In: *Proceedings of ACSAC*, ACM Press 261–269.

Hillston, J. (1996) *A Compositional Approach to Performance Modelling*, Cambridge University Press.

Kocher, P. C. (1996) Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Proceedings of CRYPTO'96. *Springer Lecture Notes in Computer Science* **1109** 104–113.

Kocher, P. C., Jaffe, J. and Jun, B. (1999) Differential power analysis. In:Proceedings of CRYPTO. *Springer Lecture Notes in Computer Science* **1666** 388–397.

Köpf, B. and Rybalchenko, A. (2010) Approximation and randomization for quantitative information-flow analysis. In: *Proceedings of CSF*, IEEE Computer Society Press 3–14.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press

Kuich, W. and Salomaa, A. (1986) *Semirings, Automata, Languages*, Monographs in Theoretical Computer Science, volume 5, Springer.

Lowe, G. (2002) Quantifying information flow. In: *Proceedings of CSFW*, IEEE Computer Society Press 18–31.

Mu, C. (2009) Measuring information flow in reactive processes. In: Proceedings of ICICS. *Springer Lecture Notes in Computer Science* **5927** 211–225.

Mu, C. and Clark, D. (2009a) An interval-based abstraction for quantifying information flow. *Electronic Notes in Theoretical Computer Science* **253** (3) 119–141.

Mu, C. and Clark, D. (2009b) Quantitative analysis of secure information flow via probabilistic semantics. In: *Proceedings of ARES*, IEEE Computer Society Press 49–57.

Rutten, J. J. M. M. (2003) Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science* **308** (1-3) 1–53.

Sabelfeld, A. and Myers, A. C. (2003) Language-based information-flow security. *IEEE Journal on Selelcted Areas in Communications* **21** (1).

Sabelfeld, A. and Sands, D. (2005) Dimensions and principles of declassification. In: *Proceedings of CSFW*, IEEE Computer Society Press 255–269.

Wittbold, J. T. and Johnson, D. M. (1990) Information flow in nondeterministic systems. In: *Proceedings of Symposium on Security and Privacy*, IEEE Computer Society Press 144–161.

https://doi.org/10.1017/S0960129513000601 Published online by Cambridge University Press