Bisimulation for demonic schedulers

Konstantinos Chatzikokolakis[1]*, Gethin Norman[2], and David Parker[2]

[1] Eindhoven University of Technology
[2] Oxford Computing Laboratory

Abstract. Bisimulation between processes has been proven a successful method for formalizing security properties. We argue that in certain cases, a scheduler that has full information on the process and collaborates with the attacker can allow him to distinguish two processes even though they are bisimilar. This phenomenon is related to the issue that bisimilarity is not preserved by refinement. As a solution, we introduce a finer variant of bisimulation in which processes are required to simulate each other under the "same" scheduler. We formalize this notion in a variant of CCS with explicit schedulers and show that this new bisimilarity can be characterized by a refinement-preserving traditional bisimilarity. Using a third characterization of this equivalence, we show how to verify it for finite systems. We then apply the new equivalence to anonymity and show that it implies strong probabilistic anonymity, while the traditional bisimulation does not. Finally, to illustrate the usefulness of our approach, we perform a compositional analysis of the Dining Cryptographers with a non-deterministic order of announcements and for an arbitrary number of cryptographers.

1 Introduction

Process algebra provides natural models for security protocols, in which nondeterminism plays an essential role, allowing to abstract from implementation details ([1–3]). In this setting, security properties are often expressed as equivalence between processes, with bisimulation being one of the most commonly used. Its application takes two distinct forms. In the first one, a protocol P is shown to be bisimilar to a specification *Spec* which can, it turn, be shown to satisfy the required security property. From this, we conclude that the protocol behaves in an equivalent way, and thus it also satisfies the property. An example is the formalization of *authenticity* in [2], in which Alice sends a message m and Bob wants to ensure that it receives m and not a different message sent by some other agent. In the specification, we allow Bob to test the received message against the real m (as if he knew it beforehand), thus the specification is obviously correct. Showing that the protocol is bisimilar to the specification, we conclude that Bob receives the correct message.

* This work was carried out while Konstantinos Chatzikokolakis was visiting Oxford University. Chatzikokolakis wishes to thank Marta Kwiatkowska for giving him the opportunity to collaborate with her group.

The second form is substantially different: we establish a bisimulation relation between two distinct instances of the protocol. From this, we conclude that the instances are *indistinguishable*, that is an attacker cannot tell the difference when observing one of them. This, in turn, means that the difference between the two instances remains hidden from the attacker. An example of this approach is the formalization of *secrecy* in [2]. If $P(m)$ is a protocol parametrized by a message m, and we show that $P(m) \sim P(m')$, where $\sim$ denotes bisimilarity, it means that the message m remains secret. Another example is the definition of *privacy* in voting protocols ([4]). The votes of Alice and Bob remain private if an instance of the protocol is bisimilar to the instance where Alice and Bob have exchanged votes.

In this paper, we focus on the second usage of bisimulation and we argue that, in the presence of a scheduler who has full view of the process, an attacker could distinguish two processes despite being bisimilar. The reason is that, in the definition of bisimulation, non-determinism is treated in a partially *angelic* way. When $P \sim Q$, the bisimulation requires that if P makes a transition α to P', Q can also make a transition α to a Q' with $Q \sim Q'$ (and vice-versa). In this definition, there are two implicit quantifiers, the second one being *existential*:

$$\begin{array}{rll} \textit{for all} \text{ transitions} & P \xrightarrow{\alpha} P' & \\ \textit{there exists} \text{ a transition} & Q \xrightarrow{\alpha} Q' & \quad \text{s.t. } P' \sim Q' \end{array}$$

In other words, Q is not forced to simulate P, it only has the possibility to do it. If we want P, Q to remain indistinguishable in the actual execution, we have to count on the additional fact that the scheduler of Q will guide it in a way that simulates P, that is the scheduler acts in favour of the process. This is, however, in contrast to the traditional idea in security that the scheduler collaborates with the attacker. If the scheduler in the implementation of Q chooses to do something different, it can allow the attacker to distinguish the two processes.

Consider the following simple example: an agent broadcasts a message m on a network which is received by the agents A and B. Each receiver then acknowledges the reception, including his identity in the acknowledgement. We could model this protocol as:

$$A = c(x).a \qquad B = c(x).b \qquad P(m) = (\nu c)(\bar{c}\langle m\rangle.\bar{c}\langle m\rangle \mid A \mid B)$$

Clearly, $P(m) \sim P(m')$, but does m remain secret? Both instances can perform two visible actions, a and b, the order of which is chosen non-deterministically. The indistinguishability of the two processes relies on the fact that the scheduler of $P(m')$ will try to simulate $P(m)$ by choosing the same order for the visible actions. If, however, the scheduler for $P(m)$ chooses a different order that the one of $P(m')$, then we can distinguish m from m' based on the output of the protocol. This could be the case, for example, if an operation is performed upon reception whose execution time is message dependent.

This consequence of angelic non-determinism can be also formulated in terms of *refinement.* A process Q refines P if it contains “less” non-determinism. This can be formalized in different ways, two commonly used ones being trace inclusion and simulation. In this paper we adopt the latter: Q refines P if P simulates Q. Then, the implementation of a protocol P is a refinement of it. However, bisimulation is not preserved by this type of refinement, which means that our security property might no longer hold in the refined version, an issue that is sometimes called the "refinement paradox" ([5, 6]). In our previous example, $P(m)$ can be refined as $(\nu c)(\bar{c}\langle m\rangle.\bar{c}\langle m\rangle \mid c(x).a.c(x).b)$ and $P(m')$ as $(\nu c)(\bar{c}\langle m'\rangle.\bar{c}\langle m'\rangle \mid c(x).b.c(x).a)$ leading to processes that can be distinguished.

Note that, in the first use of bisimulation, based on specifications, this issue does not appear. The refinement P' of a process P might not be bisimilar to the specification S but this is not a problem since we are not interested in indistinguishability. On the other hand, P' will also be a refinement of S (since $P \sim S$) which is usually enough to guarantee the required security property.

Another problem with the angelic use of non-determinism appears in probabilistic protocols. In this case, instead of traces we speak about distributions over traces and security properties are defined on these distributions. For example, strong probabilistic anonymity ([7]) requires the probability of any trace to be the same under any sender. When non-determinism and randomization are combined, trace distributions are defined after fixing a scheduler. Then, the natural extension of such definitions is to quantify over all schedulers. However, as we see in Section 6, a bisimulation between two senders does not imply strong anonymity, and the reason is the angelic use of non-determinism.

It should be noted, however, that classical bisimulation does offer some control over non-determinism. Since it is closed under contexts, bisimilar processes remain bisimilar when put in any environment. A context can act as a scheduler by restricting the set of allowed actions. Thus, bisimulation is robust against schedulers that can be expressed as contexts. However, a context cannot control the internal choices of a process, like the selection of the first receiver in the above example. Of course, we could change the process to make this choice external. However, in a big protocol with internal communication this becomes challenging. Moreover, unless we make all choices external, the remaining non-determinism will be treated in an angelic way, causing the problem with probabilistic definitions explained above. Finally, in the context approach, it is not possible to give some information to the scheduler, without making the corresponding action visible, that is, without revealing it to an observer. This could be useful, for example, to verify a protocol in which the scheduler, even if he knows some secret information, has no possibility to communicate it to the outside.

In this paper we propose a different approach. We introduce a variant of bisimulation in which the non-determinism is treated in a purely demonic way. In principle, we would like to turn the existential quantifier into a universal one. However, this is too restrictive and the resulting relation would not even be reflexive. However, we can require Q to simulate an action α of P, not under *any* scheduler but under the *same* scheduler that produced α:

$$\begin{array}{ll} \textit{for all schedulers } S\textit{, if} & P \xrightarrow{\alpha} P' \\ \textit{then under the same scheduler } S\textit{:} & Q \xrightarrow{\alpha} Q' \qquad \text{with } P' \sim Q' \end{array}$$

Note that, in general, it is not possible to speak about the "same" scheduler for two processes, since different processes have different choices. Still, this is reasonable if the processes have a similar structure (which is the case when we compare $P(m)$ to $P(m')$), in this paper we give a framework that allows to formalize this concept. The idea in the above definition that we can choose each scheduler that can break our property, however we are testing both processes under the same one. This requirement is both realistic, as we are interested in the indistinguishability of two processes when put in the same environment, and strong, since it leaves no angelic non-determinism.

To formalize this definition, that we call *demonic bisimulation*, we use a variant of probabilistic CCS with explicit schedulers, which was introduced in [8] to study the information that a scheduler has about the state of a process. This calculus allows us to speak of schedulers independently from processes, leading to a natural definition of demonic bisimulation. Then, we discuss how we can view a scheduler as a refinement operator that restricts the non-determinism of a process. We define a refinement operator, based on schedulers, and we show that demonic bisimilarity can be characterized as a refinement-preserving classical bisimilarity, for this type of refinement. Afterwards, we give a third characterization of demonic bisimilarity, that allows us to obtain an algorithm to verify it for finite processes. Finally, we apply the demonic bisimulation to the analysis of anonymity protocols. We show that demonic bisimulation, in contrast to the classical one, implies strong probabilistic anonymity. This enables us to perform a compositional analysis of the Dining Cryptographers protocol with a non-deterministic order of announcements, showing that it satisfies anonymity for an arbitrary number of cryptographers.

2 Preliminaries

In this section we recall some notions about probabilistic automata and CCS.

Probabilistic automata ([9]) A *discrete probability measure* over a set X is a function $\mu : 2^X \mapsto [0,1]$ such that $\mu(X) = 1$ and $\mu(\cup_i X_i) = \sum_i \mu(X_i)$ where X_i is a countable family of pairwise disjoint subsets of X. The set of all discrete probability measures over X will be denoted by $Disc(X)$. We will denote by $\delta(x), x \in X$ (called the *Dirac measure* on x) the probability measure that assigns probability 1 to $\{x\}$. We will also denote by $\sum_i [p_i]\mu_i$ the probability measure obtained as a convex sum of the measures μ_i.

A *simple probabilistic automaton* is a tuple $(S, q, A, \mathcal{D})$ where S is a set of states, $q \in S$ is the *initial state*, A is a set of actions and $\mathcal{D} \subseteq S \times A \times Disc(S)$ is a *transition relation.* Intuitively, if $(s, a, \mu) \in \mathcal{D}$, also written $s \xrightarrow{a} \mu$, then

$$\text{ACT}\quad \frac{}{\alpha.P \xrightarrow{\alpha}_c \delta(P)} \qquad\qquad \text{RES}\quad \frac{P \xrightarrow{\alpha}_c \mu \qquad \alpha \neq a, \bar{a}}{(\nu a)P \xrightarrow{\alpha}_c (\nu a)\mu}$$

$$\text{SUM1}\quad \frac{P \xrightarrow{\alpha}_c \mu}{P + Q \xrightarrow{\alpha}_c \mu} \qquad\qquad \text{PAR1}\quad \frac{P \xrightarrow{\alpha}_c \mu}{P \mid Q \xrightarrow{\alpha}_c \mu \mid Q}$$

$$\text{COM}\quad \frac{P \xrightarrow{a}_c \delta(P') \quad Q \xrightarrow{\bar{a}}_c \delta(Q')}{P \mid Q \xrightarrow{\tau}_c \delta(P' \mid Q')} \qquad\qquad \text{REP}\quad \frac{}{!a.P \xrightarrow{a}_c \delta(P \mid !a.P)}$$

$$\text{PROB}\quad \frac{}{\sum_i p_i P_i \xrightarrow{\tau}_c \sum_i [p_i]\delta(P_i)}$$

Fig. 1. The semantics of CCS$_p$. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

there is a transition from the state s performing the action a and leading to a distribution μ over the states of the automaton. A probabilistic automaton M is *fully probabilistic* if from each state of M there is at most one transition available. An execution α of a probabilistic automaton is a (possibly infinite) sequence $s_0 a_1 s_1 a_2 s_2 \ldots$ of alternating states and actions, such that $q = s_0$, and for each $i : s_i \xrightarrow{a_{i+1}} \mu_i$ and $\mu_i(s_{i+1}) > 0$. A *scheduler* of a probabilistic automaton $M = (S, q, A, \mathcal{D})$ is a function $\zeta : exec^*(M) \mapsto \mathcal{D}$ where $exec^*(M)$ is the set of finite executions of M, such that $\zeta(\alpha) = (s, a, \mu) \in \mathcal{D}$ implies that s is the last state of α. The idea is that a scheduler selects a transition among the ones available in $\mathcal{D}$ and it can base its decision on the history of the execution. A scheduler induces a probability space on the set of executions of M.

If $\mathcal{R}$ is a relation over a set S, then we can lift the relation to probability distributions over S using the standard weighting function technique (see [9] for details). If is an equivalence relation then the lifting can be simplified: $\mu_1\ \mathcal{R}\ \mu_2$ iff for all equivalence classes $\mathcal{E} \in S/\mathcal{R}$, $\mu_1(\mathcal{E}) = \mu_2(\mathcal{E})$. We can now define simulation and bisimulation for simple probabilistic automata.

Definition 1. *Let $(S, q, A, \mathcal{D})$ be a probabilistic automaton. A relation $\mathcal{R} \subseteq S \times S$ is a* simulation *iff for all $(s_1, s_2) \in \mathcal{R}, a \in A$: if $s_1 \xrightarrow{a} \mu_1$ then there exists μ_2 such that $s_2 \xrightarrow{a} \mu_2$ and $\mu_1\ \mathcal{R}\ \mu_2$. A simulation $\mathcal{R}$ is a* bisimulation *if it is also symmetric (thus, it is an equivalence relation). We define $\sqsubseteq, \sim$ as the largest simulation and bisimulation on S respectively.*

CCS with internal probabilistic choice Let a range over a countable set of *channel names.* The syntax of CCS$_p$ is:

$$\begin{array}{rcll}
\alpha & ::= & a \mid \bar{a} \mid \tau & \textbf{prefixes} \\
P, Q & ::= & a.P \mid P \mid Q \mid P + Q \mid \sum_i p_i P_i \mid (\nu a)P \mid !a.P \mid 0 & \textbf{processes}
\end{array}$$

The term $\sum_i p_i P_i$ represents an *internal probabilistic choice*, all the remaining operators are from standard CCS. We will also use the notation $P_1 +_p P_2$ to represent a binary sum $\sum_i p_i P_i$ with $p_1 = p$ and $p_2 = 1 - p$. Finally, we use

$I ::= 0\,I \mid 1\,I \mid \epsilon$	**label indexes**
$L ::= l^I$	**labels**
$P, Q ::=$	**processes**
$L{:}\alpha.P$	prefix
$\mid P \mid Q$	parallel
$\mid P + Q$	nondeterm. choice
$\mid L{:}\sum_i p_i P_i$	internal prob. choice
$\mid (\nu a)P$	restriction
$\mid !L{:}a.P$	replicated input
$\mid L{:}0$	nil

$S, T ::=$	**scheduler**
$L.S$	schedule single action
$\mid (L, L).S$	synchronization
$\mid$ **if** L	label test
then S	
else S	
$\mid 0$	nil
$CP ::= P \parallel S$	**complete process**

Fig. 2. The syntax of CCS$_\sigma$

replicated input instead of replication or recursion, as this simplifies the presentation. Note that replication in CCS is not equally powerful as recursion, but their difference is orthogonal to the goals of this paper. The semantics of a CCS$_p$ term is a probabilistic automaton defined according to the rules in Figure 1. We use $\longrightarrow_c$ to distinguish the transitions of classical CCS from other transition systems defined later in the paper. We also denote by $\mu \mid Q$ the measure μ' such that $\mu'(P \mid Q) = \mu(P)$ for all processes P and $\mu'(R) = 0$ if R is not of the form $P \mid Q$, and similarly for $(\nu a)\mu$.

3 A variant of CCS$_p$ with explicit scheduler

In this section we present a variant of CCS$_p$ in which the scheduler is explicit, in the sense that it has a specific syntax and its behaviour is defined by the operational semantics of the calculus. This calculus was proposed in [8]; we will refer to it as CCS$_\sigma$. Processes in CCS$_\sigma$ contain labels that allow us to refer to a particular sub-process. A scheduler also behaves like a process, using however a different and much simpler syntax, and its purpose is to guide the execution of the main process using the labels that the latter provides.

3.1 Syntax

Let a range over a countable set of *channel names* and l over a countable set of *atomic labels*. The syntax of CCS$_\sigma$, shown in Figure 2, is the same as the one of CCS$_p$ except for the presence of labels. These are used to select the subprocess which "performs" a transition. Since only the operators with an initial rule can originate a transition, we only need to assign labels to the prefix and to the probabilistic sum. We use labels of the form l^s where l is an atomic label and the index s is a finite string of 0 and 1, possibly empty. Indexes are used to avoid multiple copies of the same label in case of replication. As explained in the semantics, each time a process is replicated we relabel it using appropriate indexes. To simplify the notation, we use base labels of the form $l_1, \ldots, l_n$, and we write ${}^i a.P$ for $l_i{:}a.P$.

$$\text{ACT}\ \frac{}{l{:}\alpha.P \parallel l.S \xrightarrow{\alpha}_s \delta(P \parallel S)} \qquad \text{RES}\ \frac{P \parallel S_l \xrightarrow{\alpha}_s \mu \quad \alpha \neq a, \overline{a}}{(\nu a)P \parallel S_l \xrightarrow{\alpha}_s (\nu a)\mu}$$

$$\text{SUM1}\ \frac{P \parallel S_l \xrightarrow{\alpha}_s \mu}{P + Q \parallel S_l \xrightarrow{\alpha}_s \mu} \qquad \text{PAR1}\ \frac{P \parallel S_l \xrightarrow{\alpha}_s \mu}{P \mid Q \parallel S_l \xrightarrow{\alpha}_s \mu \mid Q}$$

$$\text{COM}\ \frac{P \parallel l_1 \xrightarrow{a}_s \delta(P' \parallel 0) \qquad Q \parallel l_2 \xrightarrow{\overline{a}}_s \delta(Q' \parallel 0)}{P \mid Q \parallel (l_1, l_2).S \xrightarrow{\tau}_s \delta(P' \mid Q' \parallel S)}$$

$$\text{PROB}\ \frac{}{l{:}\sum_i p_i P_i \parallel l.S \xrightarrow{\tau}_s \sum_i p_i \delta(P_i \parallel S)} \qquad \text{REP}\ \frac{}{!l{:}a.P \parallel l.S \xrightarrow{\alpha}_s \delta(\rho_0 P \mid !l{:}a.\rho_1 P \parallel S)}$$

$$\text{IF1}\ \frac{l \in tl(P) \qquad P \parallel S_1 \xrightarrow{\alpha}_s \mu}{P \parallel \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \xrightarrow{\alpha}_s \mu} \qquad \text{IF2}\ \frac{l \notin tl(P) \qquad P \parallel S_2 \xrightarrow{\alpha}_s \mu}{P \parallel \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \xrightarrow{\alpha}_s \mu}$$

Fig. 3. The semantics of complete CCS$_\sigma$ processes. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

A scheduler selects a sub-process for execution on the basis of its label, so we use $l.S$ to represent a scheduler that selects the process with label l and continues as S. In the case of synchronization we need to select two processes simultaneously, hence we need a scheduler of the form $(l_1, l_2).S$. We will use S_l to denote a scheduler of one of these forms (that is, a scheduler that starts with a label or pair of labels). The **if-then-else** construct allows the scheduler to test whether a label is available in the process (in the top-level) and act accordingly. A complete process is a process put in parallel with a scheduler, for example $l_1{:}a.l_2{:}b \parallel l_1.l_2$. We define $\mathcal{P}, \mathcal{CP}$ to be the sets of all processes and all complete CCS$_\sigma$ processes respectively. Note that for processes with an infinite execution path we need schedulers of infinite length. So, to be formally correct, we should define schedulers as infinite trees, instead of using a BNF grammar.

3.2 Semantics for complete processes

The semantics of CCS$_\sigma$ is given in terms of a probabilistic automaton whose state space is $\mathcal{CP}$ and whose transitions are given by the rules in Figure 3. We denote the transitions by $\longrightarrow_s$ to distinguish it from other transition systems.

ACT is the basic communication rule. In order for $l{:}\alpha.P$ to perform α, the scheduler should select this process for execution, so the scheduler needs to be of the form $l.S$. After this execution the complete process will continue as $P \parallel S$. The RES rule models restriction on channel a: communication on this channel is not allowed by the restricted process. We denote by $(\nu a)\mu$ the measure μ' such that $\mu'((\nu a)P \parallel S) = \mu(P \parallel S)$ for all processes P and $\mu'(R \parallel S) = 0$ if R is not of the form $(\nu a)P$. SUM1 models nondeterministic choice. If $P \parallel S$ can perform a transition to μ, which means that S selects one of the labels of P, then $P + Q \parallel S$ will perform the same transition, i.e. the branch P of the choice will be selected and Q will be discarded. For example:

$$l_1{:}a.P + l_2{:}b.Q \parallel l_1.S \xrightarrow{a}_s \delta(P \parallel S)$$

Note that the operands of the sum do not have labels, the labels belong to the subprocesses of P and Q. In the case of nested choices, the scheduler must select the label of a prefix, thus resolving all the choices at once.

PAR1, modelling parallel composition, is similar: the scheduler selects P to perform a transition on the basis of the label. The difference is that in this case Q is not discarded; it remains in the continuation. $\mu \mid Q$ denotes the measure μ' such that $\mu'(P \mid Q \parallel S) = \mu(P \parallel S)$. COM models synchronization. If $P \parallel l_1$ can perform the action a and $Q \parallel l_2$ can perform $\bar{a}$, then $(l_1, l_2).S$ can synchronize the two by scheduling both l_1 and l_2 at the same time. PROB models internal probabilistic choice. Note that the scheduler cannot affect the outcome of the choice, it can only schedule the choice as a whole (this is why a probabilistic sum has a label) and the process will move to a measure containing all the operands with corresponding probabilities.

REP models replicated input. This rule is the same as in CCS, with the addition of a re-labeling operator ρ_i. The reason for this is that we want to avoid ending up with multiple copies of the same label as the result of replication, since this would create ambiguities in scheduling as explained in Section 3.3. $\rho_i P$ appends $i \in \{0, 1\}$ to the index of all labels of P, for example:

$$\rho_i l^s\!:\!\alpha.P = l^{si}\!:\!\alpha.\rho_i P$$

and similarly for the other operators. Note that we relabel only the resulting process, not the continuation of the scheduler: there is no need for relabeling the scheduler since we are free to choose the continuation as we please.

Finally **if-then-else** allows the scheduler to adjust its behaviour based on the labels that are available in P. $tl(P)$ gives the set of top-level labels of P and is defined as:

$$tl(l\!:\!\alpha.P) = tl(l\!:\!\textstyle\sum_i p_i P_i) = tl(!l\!:\!a.P) = tl(l\!:\!0) = \{l\}$$

and as the union of the top-level labels of all sub-processes for the other operators. Then **if** l **then** S_1 **else** S_2 behaves like S_1 if l is available in P and as S_2 otherwise.

A process is blocked if it cannot perform a transition under any scheduler. A scheduler S is *non-blocking* for a process P if it always schedules some transition, except when P itself is blocked.

3.3 Deterministic labelings

The idea in CCS$_\sigma$ is that a *syntactic* scheduler will be able to completely resolve the nondeterminism of the process, without needing to rely on a *semantic* scheduler at the level of the automaton. To achieve this we impose a condition on the labels of CCS$_\sigma$ processes. A *labeling* for P is an assignment of labels to the subprocesses of P that require a label. A labeling for P is *deterministic* iff for all schedulers S there is at most one transition of $P \parallel S$ enabled at any time, in other words the corresponding automaton is fully probabilistic. In the rest of the paper, we only consider processes with deterministic labelings.

A simple case of deterministic labelings are the *linear* ones, containing pairwise distinct labels (a more precise definition of linear labelings requires an extra condition and can be found in [10]). It can be shown that linear labelings are preserved by transitions and are deterministic. However, the interesting case is that we can construct labelings that are deterministic without being linear. The usefulness of non-linear labelings is that they limit the power of the scheduler, since the labels provide information about the current state and allow the scheduler to choose different strategies through the use of **if-then-else**. Consider, for example, the following process whose labeling is deterministic but not linear:

$$l\!:\!({}^1\bar{a}.R_1 +_p {}^1\bar{a}.R_2) \mid {}^2a.P \mid {}^3a.Q \tag{1}$$

Since both branches of the probabilistic sum have the same label l_1, the scheduler cannot resolve the choice between P and Q based on the outcome of the probabilistic choice. Another use of non-linear labeling is the encoding of "private" value passing:

$$l\!:\!c(x).P \;\triangleq\; \textstyle\sum_i l\!:\!cv_i.P[v_i/x] \qquad\qquad l\!:\!\bar{c}\langle v\rangle.P \;\triangleq\; l\!:\!\overline{cv}.P$$

This is the usual encoding of value passing in CCS except that we use the same label in all the branches of the nondeterministic sum. To ensure that the resulting labeling is deterministic we should restrict the channels cv_i and make sure that there is at most one output on c. For instance, the labeling of the process $(\nu c)(l_1\!:\!c(x).P \mid l_2\!:\!\bar{c}\langle v_1\rangle)$ is deterministic. This way, the reception of a message is visible to the scheduler, but not the received value.

4 Demonic Bisimulation

In the introduction, we showed that classical bisimulation treats non-determinism in a partially angelic way. As a consequence, a scheduler that has full control over a process can be used to distinguish two bisimilar processes. In this section, we define a strict variant of bisimulation, called demonic bisimulation, which treats non-determinism in a purely demonic way. We first define this equivalence in terms of schedulers. Then, we change perspective and we view a scheduler as a refinement operator. We show that demonic bisimilarity can be characterized as a classical bisimilarity preserved under all refinements.

4.1 Definition using schedulers

An informal definition of demonic bisimulation was already given in the introduction. P is demonic-bisimilar to Q, written $P \sim_D Q$ if:

$$\begin{array}{rl} \textit{for all schedulers } S\textit{, if} & P \xrightarrow{\alpha} P' \\ \textit{then under the same scheduler } S\textit{:} & Q \xrightarrow{\alpha} Q' \qquad \text{with } P' \sim_D Q' \end{array}$$

To define demonic bisimulation concretely, we need a framework that allows a single scheduler to be used with different processes. CCS$_\sigma$ does exactly this: it gives semantics to $P \parallel S$ for any process P and scheduler S (of course, S might be blocking for some processes and non-blocking for others).

If μ is a discrete measure on $\mathcal{P}$, we denote by $\mu \parallel S$ the discrete measure μ' on $\mathcal{CP}$ such that $\mu'(P \parallel S) = \mu(P)$ for all $P \in \mathcal{P}$ and $\mu'(P \parallel S') = 0$ for all $S' \neq S$ (note that all transition rules of Fig. 3 produce measures of this form). We can now give a concrete definition of demonic bisimulation.

Definition 2 (Demonic bisimulation). *An equivalence relation $\mathcal{R}$ on $\mathcal{P}$ is a* demonic bisimulation *iff for all $(P_1, P_2) \in \mathcal{R}, a \in A$ and all schedulers S: if S is non-blocking for P_1 and $P_1 \parallel S \xrightarrow{\alpha} \mu_1 \parallel S'$ then the same scheduler S is non-blocking for P_2 and $P_2 \parallel S \xrightarrow{\alpha} \mu_2 \parallel S'$ with $\mu_1 \mathrel{\mathcal{R}} \mu_2$. We define demonic bisimlarity $\sim_D$ as the largest demonic bisimulation on $\mathcal{P}$.*

Consider again the example of the introduction. We define:

$$A = {}^1c(x).{}^2a \qquad B = {}^3c(x).{}^4b \qquad P(m) = (\nu c)({}^5\bar{c}\langle m\rangle.{}^6\bar{c}\langle m\rangle \mid A \mid B)$$

Note that $P(m), P(m')$ share the same labels. This choice of labels states that whenever a scheduler chooses an action in $P(m)$, it has to schedule the same action in $P(m')$. Then it is easy to see that $P(m) \sim_D P(m')$. A scheduler that selects A first in $P(m)$ will also select A first in $P(m')$, leading to the same order of actions. Under this definition, we do not rely on angelic non-determinism for $P(m')$ to simulate $P(m)$, we have constructed our model in a way that forces a scheduler to perform the same action in both processes. Note that we could also choose to put different labels in $\bar{c}\langle m\rangle, \bar{c}\langle m'\rangle$, hence allowing them to be scheduled in a different way. In this case $\sim_D$ will no longer hold, exactly because we can now distinguish the two processes using an **if-then-else** schedule who depends on the message.

As a usual sanity check, we show that $\sim_D$ is a congruence.

Proposition 1. *$\sim_D$ is closed under contexts.*

4.2 Characterization using refinement

Another way of looking at schedulers is in terms of refinement. As discussed in the introduction, a process Q refines P if it contains "less" non-determinism. A typical definition is in terms of simulation: Q refines P if $Q \sqsubseteq P$. For example, a is a refinement of $a + b$ where the non-deterministic choice has been resolved. Thus, a scheduler can be seen as a way to refine a process by resolving the non-determinism. For example, $l_1 : a$ can be seen as the refinement of $l_1 : a + l_2 : b$ under the scheduler l_1. Moreover, partial schedulers can be considered as resolving only part of the non-determinism. For example, for the process ${}^1a.({}^3c + {}^4d) + {}^2b$, the scheduler l_1 will resolve the first non-deterministic choice but not the second.

It has been observed that many security properties are not preserved by refinement, a fact that is sometimes called the "refinement paradox". If we define security properties using bisimulation this issue immediately arises. For example, $a|b$ is bisimilar to $a.b + b.a$ but if we refine them to $a.b$ and $b.a$ respectively, they

$$\varphi_0(P) = P \tag{2}$$
$$\varphi_{\lambda.S}(\lambda{:}\alpha.P) = \lambda{:}\alpha.\varphi_S(P) \tag{3}$$
$$\varphi_{\lambda.S}(P+Q) = \varphi_{\lambda.S}(P) + \varphi_{\lambda.S}(Q) \tag{4}$$
$$\varphi_{\lambda.S}((\nu a)P) = (\nu a)\varphi_{\lambda.S}(P) \tag{5}$$
$$\varphi_{l.S}(l{:}\textstyle\sum_i p_i P_i) = l{:}\sum_i p_i \varphi_S(P_i) \tag{6}$$
$$\varphi_{\lambda.S}(P \mid Q) = \begin{cases} \lambda{:}\alpha.\varphi_S(P' \mid Q) & \text{if } \varphi_\lambda(P) \xrightarrow{\alpha}_c \delta(P') \\ \lambda{:}\sum_i p_i \varphi_S(P_i \mid Q) & \text{if } \varphi_\lambda(P) \xrightarrow{\tau}_c \sum_i [p_i]\delta(P_i) \\ \lambda{:}\tau.\varphi_S(P' \mid Q') & \text{if } \lambda = (l_1, l_2) \text{ and} \\ & \varphi_{l_1}(P) \xrightarrow{a}_c \delta(P'), \varphi_{l_2}(Q) \xrightarrow{\overline{a}}_c \delta(Q') \end{cases} \tag{7}$$
$$\varphi_{l.S}(!l{:}a.P) = l{:}a.\varphi_S(\rho_0 P \mid !l{:}\ a.\rho_1 P) \tag{8}$$
$$\varphi_S(P) = \begin{cases} \varphi_{S_1}(P) & \text{if } l \in tl(P) \text{ where } S = \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \\ \varphi_{S_2}(P) & \text{if } l \notin tl(P) \end{cases} \tag{9}$$
$$\varphi_S(P) = 0 \quad \text{if none of the above is applicable (e.g. } \varphi_{l_1}(l_2{:}\alpha.P) = 0) \tag{10}$$

Fig. 4. Refinement of CCS$_\sigma$ processes under a scheduler. The symmetric cases for the parallel operator have been omitted for simplicity

are no longer bisimilar. Clearly, if we want to preserve bisimilarity we have to refine both processes in a consistent way. In this section, we introduce a refinement operator based on schedulers. We are then interested in processes that are not only bisimilar, but also preserve bisimilarity under this refinement. We show that this stronger equivalence coincides with demonic bisimilarity.

With a slight abuse of notation we extend the transitions $\longrightarrow_c$, that is the traditional transition system for CCS, to CCS$_\sigma$ processes, by simply ignoring the labels, which are then only used for the refinement. Let S be a finite scheduler and P a CCS$_\sigma$ process. The refinement of P under S, denoted by $\varphi_S(P)$, is a new CCS$_\sigma$ process. The function $\varphi_S : \mathcal{P} \to \mathcal{P}$ is defined in Figure 4. Note that φ_S does not perform transitions, it only blocks the transitions that are not enabled by S. Thus, it reduces the non-determinism of the process. The scheduler might be partial: a scheduler 0 leaves the process unaffected (2). Thus, the resulting process might still have non-deterministic choices. A prefix is allowed in the refined process only if its label is selected by the scheduler (3), otherwise the refined process is equal to 0 (10). Case (4) applies the refinement to both operands. Note that, if the labeling is deterministic, at most one of the two will have transitions enabled. The most interesting case is the parallel operator (7). There are three possible executions for $P \mid Q$. An execution of P alone, of Q alone or a synchronization between the two. The refined version enforces the one selected by the scheduler (the symmetric cases have been omitted for simplicity). This is achieved by explicitly prefixing the selected action, for example $l_1{:}a \mid l_2{:}b$ refined by l_1 becomes $l_1{:}a.(0 \mid l_2{:}b)$. If P performs a probabilistic choice, then we have to use a probabilistic sum instead of an action prefix. The case of $!P$ (8) is similar to the prefix (3) and the rest of the cases are self-explanatory.

The intention behind the definition of ϕ_S is to refine CCS$_\sigma$ processes. $\phi_S(P)$ cointains only the choices of P that are selected by the scheduler S. We now show that the result is indeed a refinement:

Proposition 2. *For all CCS$_\sigma$ processes P and schedulers S: $\phi_S(P) \sqsubseteq P$*

Note that $\sqsubseteq$ is the simulation relation on $\mathcal{P}$ wrt the classical CCS semantics $\longrightarrow_c$. Also, let $\sim$ be the bisimilarity relation on $\mathcal{P}$ wrt $\longrightarrow_c$. A nice property of this type of refinement is that it allows to refine two processes in a consistent way. This enables us to define a refinement-preserving bisimulation.

Definition 3. *An equivalence relation $\mathcal{R}$ on $\mathcal{P}$ is an* R-bisimulation *iff for all $(P_1, P_2) \in \mathcal{R}$ and all finite schedulers S: $\varphi_S(P_1) \sim \varphi_S(P_2)$. We denote by $\sim_R$ the largest R-bisimulation.*

Note that $P_1 \sim_R P_2$ implies $P_1 \sim P_2$ (for $S = 0$). Moreover $\sim_R$ requires P_1, P_2 to remain bisimilar under any refinement. Finally, we show that processes that preserve bisimilarity under this type of refinement are exactly the ones that are demonic-bisimilar.

Theorem 1. *The equivalence relations $\sim_R$ and $\sim_D$ coincide.*

5 Verifying demonic bisimilarity for finite processes

In the previous section we gave two characterizations of demonic bisimilarity (Def. 2 and 3). They both, however, have the drawback of quantifying over all schedulers. This makes the verification of the equivalence difficult, even for finite state processes. To overcome this difficulty, we give a third characterization of demonic bisimilarity. This one, is based on traditional bisimilarity on a modified transition system, where labels annotate the performed actions. We then use this characterization to adapt an algorithm for verifying probabilistic bisimilarity to our settings.

5.1 Characterization using a modified transition system

In this section we give a modified semantics for CCS$_\sigma$ processes without schedulers. The semantics are given by means of a simple probabilistic automaton with state space $\mathcal{P}$, displayed in Figure 5 and denoted by $\longrightarrow_a$ to distinguished from the transition system previously defined. The difference is that now the labels annotate the actions instead of being used by the scheduler. Thus, we have actions of the form $\lambda{:}\alpha$ where λ is l or (l_1, l_2), and α is a channel, an output on a channel or τ. Note that, in the case of synchronization (COM), we combine the labels l_1, l_2 of the actions $a, \overline{a}$ and we annotate the resulting τ action by (l_1, l_2). All rules match the corresponding transitions for complete processes. Since no schedulers are involved here, the rules IF1 and IF2 are completely removed.

$$\text{ACT}\ \frac{}{l{:}\alpha.P \xrightarrow{l:\alpha}_a \delta(P)} \qquad \text{RES}\ \frac{P \xrightarrow{l:\alpha}_a \mu \quad \alpha \neq a, \overline{a}}{(\nu a)P \xrightarrow{l:\alpha}_a (\nu a)\mu}$$

$$\text{SUM1}\ \frac{P \xrightarrow{l:\alpha}_a \mu}{P + Q \xrightarrow{l:\alpha}_a \mu} \qquad \text{PAR1}\ \frac{P \xrightarrow{l:\alpha}_a \mu}{P \mid Q \xrightarrow{l:\alpha}_a \mu \mid Q}$$

$$\text{COM}\ \frac{P \xrightarrow{l_1:a}_a \delta(P') \quad Q \xrightarrow{l_2:\overline{a}}_a \delta(Q')}{P \mid Q \xrightarrow{(l_1,l_2):\tau}_a \delta(P' \mid Q')} \qquad \text{PROB}\ \frac{}{l{:}\sum_i p_i P_i \xrightarrow{l:\tau}_a \sum_i p_i \delta(P_i)}$$

$$\text{REP}\ \frac{}{!l{:}a.P \xrightarrow{l:a}_a \delta(\rho_0 P \mid !l{:}a.\rho_1 P)}$$

Fig. 5. Semantics for CCS$_\sigma$ processes without schedulers. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

Note that, even if the labeling is deterministic, the probabilistic automaton defined by the semantics without schedulers is not necessarily fully probabilistic since multiple transitions might be enabled at the same time. However, all transitions of a process will have pairwise distinct actions.

We can now characterize demonic bisimilarity using this transition system.

Definition 4. *An equivalence relation $\mathcal{R}$ on $\mathcal{P}$ is an* A-bisimulation *iff*

i) it is a bisimulation wrt $\longrightarrow_a$, and
ii) $tl(P_1) = tl(P_2)$ for all non-blocked $P_1, P_2 \in \mathcal{R}$

We define $\sim_A$ as the largest A-bisimulation on $\mathcal{P}$.

Theorem 2. *The equivalence relations $\sim_D$ and $\sim_A$ coincide.*

Essentially, we have encoded the schedulers in the actions of the transition system $\longrightarrow_a$. Thus, if two processes perform the same action in $\longrightarrow_a$ it means that they perform the same action with the same scheduler in $\longrightarrow_s$. Note that the relation $\sim_A$ is stricter that the classical bisimilarity. This is needed because schedulers have the power to check the top-level labels of a process, even if this label is not "active", that is it does not correspond to a transition. We could modify the semantics of the **if-then-else** operator, in order to use the traditional bisimilarity in the above theorem. However, this would make the schedulers less expressive. Indeed, it can be shown ([8]) that for any semantic scheduler (that is, one defined on the automaton) of a CCS$_p$ process P, we can create a syntactic scheduler that has the same behaviour on P labeled with a linear labeling. This property, though, is lost under the modified **if-then-else**.

5.2 An algorithm for finite state processes

We can now use $\sim_A$ to verify demonic bisimilarity for finite state processes. For this, we adapt the algorithm of Baier ([11]) for probabilistic bisimilarity. This algorithm works on partitions $X = \{S_1, \ldots, S_n\}$ where $S_1, \ldots, S_n$ are pairwise disjoint subsets of the (finite) state space S. It starts from the trivial partition $X_0 = \{S\}$ and in each iteration updates the current partition X by possibly splitting each set S_i into parts. Two elements of S_i are put in different sets if there is a transition from each of them that assigns a different probability to some set $S_j \in X$. This iteration is guaranteed to terminate, and the resulting partition X_n is the set of equivalence classes of $\sim$.

In our case, the state space is the subset of $\mathcal{P}$ that it reachable from the given processes, and is assumed to be finite. Moreover, we are interested in verifying $\sim_A$ which has the additional requirement that related non-blocked processes should have the same set of top labels. One approach would be to take this requirement into account when updating the partitions in the above algorithm. However, this can actually be performed in a pre-processing step. Define an equivalence $\mathcal{R}$ as

$$(P, Q) \in \mathcal{R} \quad \text{iff} \quad tl(P) = tl(Q) \text{ or } P, Q \text{ are blocked}$$

Then we apply the above algorithm, starting from the partition $X_0 = S/\mathcal{R}$. The elements of each set $S_i \in X_0$ have the same set of top-level labels (unless they are blocked). Since each iteration only splits each set into parts, the elements of a set $S_i \in X_n$, where X_n is the resulting partition, will also have the same top-level labels. Hence, it is easy to see that X_n will be the set of equivalence classes of $\sim_A$.

Similarly to the original algorithm, its complexity is $O(n^2 \cdot m)$ where n is the number of states and m the number of transitions. An implementation of the algorithm is available at [12] and has been used to verify some of the results of the following section.

6 An application to security

In this section, we apply the demonic bisimulation to the verification of anonymity protocols. First, we formalize anonymity in terms of equivalence between different instances of the protocol. We then show that this definition implies strong probabilistic anonymity, which was defined in [7] in terms of traces. This allows us to perform an easier analysis of protocols by exploiting the algebraic properties of an equivalence. We perform such a compositional analysis on the Dining Cryptographers protocol with non-deterministic order of announcements.

6.1 Probabilistic anonymity

Consider a protocol in which a set $\mathcal{A}$ of *anonymous events* can occur. An event $a_i \in \mathcal{A}$ could mean, for example, that user i performed an action of interest, the purpose of the protocol is to keep such events hidden. The instance of the protocol when a_i occurs is modelled by the process $Prot_i$. Typically, the selection of anonymous event is performed in the beginning of the protocol, for example a user i decides to send a message, and then the protocol proceeds as $Prot_i$. Thus, we model the protocol as

$$Prot \triangleq l{:}\textstyle\sum_i p_i \; l_i{:}a_i.Prot_i \tag{11}$$

The protocol starts by selecting an anonymous event probabilistically (using any distribution, the anonymity definition is independent of it). Then, the selection is announced on the channel a_i, this serves the purpose of keeping the information about the selected event in the produced trace. Then the protocol continues as $Prot_i$, which produces the events that the attacker can observe (of course, the actions a_i are not visible to the attacker). Define $Tr(Prot)$ as the set of traces produced by $Prot$ (under any scheduler), note that they all are of the form $a_i\omega$, where ω is a sequence of visible actions of $Prot_i$. The goal of the attacker is to deduce a_i from ω, hence anonymity is provided if all anonymous events produce traces with the same probability.

Definition 5 (strong probabilistic anonymity). *A protocol Prot is anonymous iff for all schedulers S, all anonymous events $a_i, a_j \in \mathcal{A}$ and all traces $\omega : p_S(a_i\omega|\boldsymbol{a_i}) = p_S(a_j\omega|\boldsymbol{a_j})$ where p_S is the probability distribution on $Tr(Prot)$ induced by the semantics of $Prot \parallel S$, and $\boldsymbol{a_i} = \{a_i\omega | a_i\omega \in Tr(Prot)\}$.*

This is the definition of probabilistic anonymity given in [7], adapted to our setting, and is a direct generalization of the definition for the purely probabilistic case. We now give an alternative definition, based on demonic bisimulation.

Definition 6 (equivalence based anonymity). *A protocol (of the form (11)) satisfies anonymity iff for all anonymous events $a_i, a_j \in \mathcal{A} : Prot_i \sim_D Prot_j$.*

The idea behind this definition is that, if $Prot_i, Prot_j$ are demonic-bisimilar, they should behave in the same way under all schedulers, thus producing the same observation. Indeed, we can show that the above definition implies Def. 5.

Proposition 3. *If $Prot_i \sim_D Prot_j$ for all i, j then the protocol satisfies strong probabilistic anonymity (Def. 5)*

It is worth noting that, on the other hand, $Prot_i \sim Prot_j$ does not imply Def. 5, as we see in the next section.

6.2 Analysis of the Dining Cryptographers protocol

The problem of the Dining Cryptographers is the following: Three cryptographers dine together. After the dinner, the bill has to be paid by either one of them or by another agent called the master. The master decides who will pay and then informs each of them separately whether he has to pay or not. The cryptographers would like to find out whether the payer is the master or one of them. However, in the latter case, they wish to keep the payer anonymous.

The Dining Cryptographers Protocol (DCP) solves the above problem as follows: each cryptographer tosses a fair coin which is visible to himself and his neighbour to the right. Each cryptographer checks the two adjacent coins and, if he is not paying, announces *agree* if they are the same and *disagree* otherwise. However, the paying cryptographer says the opposite. It can be proved that the master is paying if and only if the number of *disagrees* is even ([13]).

We model the protocol, for the general case of a ring of n cryptographers, as shown in Figure 6. The symbols $\oplus, \otimes$ represent the addition modulo n and

$$
\begin{aligned}
\mathit{CryptP}_i &\triangleq {}^{1,i}c_i(\mathit{coin}_1).{}^{2,i}c_i(\mathit{coin}_2).{}^{3,i}\overline{\mathit{out}}_i\langle \mathit{coin}_1 \otimes \mathit{coin}_2\rangle \\
\mathit{Crypt}_i &\triangleq {}^{1,i}c_i(\mathit{coin}_1).{}^{2,i}c_i,(\mathit{coin}_2).{}^{3,i}\overline{\mathit{out}}_i\langle \mathit{coin}_1 \otimes \mathit{coin}_2 \otimes 1\rangle \\
\mathit{Coin}_i &\triangleq l_{4,i}\!:\!(({}^{5,i}\bar{c}_i\langle 0\rangle \mid {}^{6,i}\bar{c}_{i\oplus 1}\langle 0\rangle) +_{0.5} ({}^{5,i}\bar{c}_i\langle 1\rangle \mid {}^{6,i}\bar{c}_{i\oplus 1}\langle 1\rangle)) \\
\mathit{Prot}_i &\triangleq (\nu \boldsymbol{c})(\mathit{CryptP}_i \mid \textstyle\prod_{j\neq i} \mathit{Crypt}_j \mid \prod_{j=0}^{n-1} \mathit{Coin}_j)
\end{aligned}
$$

Fig. 6. Encoding of the dining cryptographers protocol

modulo 2 (xor) respectively. $\mathit{Crypt}_i, \mathit{CryptP}_i$ model the cryptographer i acting as non-payer or payer respectively. Coin_i models the i-th coin, shared between cryptographers i and $i \oplus 1$. Finally, Prot_i is the instance of the protocol when cryptographer i is the payer, and consists of CryptP_i, all other cryptographers as non-payers, and all coins. An external observer is supposed to see only the announcements $\overline{\mathit{out}}_i\langle\cdot\rangle$. As discussed in [7], DCP satisfies anonymity if we abstract from their order. If their order is observable, on the contrary, a scheduler can reveal the identity of the payer to the observer by forcing the payer to make his announcement first, or by selecting the order based on the value of the coins.

As we explained in the introduction, the angelic use of non-determinism is problematic for definitions like Def. 5 that quantify over all schedulers. For the DCP, we can show that $\mathit{Prot}_i \sim \mathit{Prot}_j$ for all i, j. However, this model does not satisfy Def. 5, because a scheduler that depends its behaviour on the value of a coin will lead to different traces.

In CCS$_\sigma$ we can be precise about the information that is revealed to the scheduler. In the encoding of Fig. 6, we have used the same labels on both sides of the probabilistic choice in Coin_i. As a consequence, after performing the choice, the scheduler cannot use an **if-then-else** to find out which was the outcome, so his decision will be independent of the coin's value. Similarly, the use of private value passing (see Section 3.3) guarantees that the scheduler will not see which value is transmitted by the coin to the cryptographers. Then we can show that for any number of cryptographers:

$$\mathit{Prot}_i \sim_D \mathit{Prot}_j \quad \forall 1 \leq i, j \leq n \tag{12}$$

For a fixed number of cryptographers, (12) can be verified automatically using the algorithm of Section (5.2). We have used a prototype implementation to verify demonic bisimilarity for a very small number of cryptographers (after that, the state space becomes too big). However, using the algebraic properties of sim_D we can perform a compositional analysis and prove (12) for any number of cryptographers. Due to the limited space, this approach is described in the appendix.

On the other hand, if we want to consider schedulers that can depend on the coins, we can choose different labels on each side of Coin_i. This would violate $\sim_D$, revealing that the protocol no longer satisfies strong anonymity.

7 Related work

Various works in the area of probabilistic automata introduce restrictions to the scheduler to avoid violating security properties ([14–16]). Their approach is based on dividing the actions of each component of the system in equivalence classes (*tasks*). The order of execution of different tasks is decided in advance by a so-called *task scheduler*. The remaining nondeterminism within a task is resolved by a second demonic schedule. In our approach, the order of execution is still decided non-deterministically by a demonic scheduler, but we impose that the scheduler will make the same decision in both processes.

Refinement operators that preserve various security properties are given in [17, 18]. In our approach, we impose that the refinement operator should preserve bisimilarity, obtaining a stronger equivalence.

In the probabilistic setting, a bisimulation that quantifies over all schedulers is used in [19]. In this work, however, the scheduler only selects the action and the remaining non-determinism is resolved probabilistically (using a uniform distribution). This avoids the problem of angelic non-determinism but weakens the power of the scheduler.

On the other hand, [20] gives an equivalence-based definition of anonymity for the Dining Cryptographers, but in a possibilistic setting. In this case the scheduler is clearly angelic, since anonymity relies on a non-deterministic selection of the coins. Our definition is the probabilistic counterpart of this work, which was problematic due to the angelic use of non-determinism.

8 Conclusion and future work

We have introduced a notion of bisimulation where processes are required to simulate each other under the same scheduler. We have characterized this equivalence in three different ways: using syntactic schedulers in a variant of CCS, using a refinement operator based on schedulers and using a modified transition system where labels annotate the actions. We have applied this notion to anonymity showing that strong anonymity can be defined in terms of equivalence, leading to a compositional analysis of the dining cryptographers with non-deterministic order of announcements.

As future work, we want to investigate the effect of angelic non-determinism to other process equivalences. Many of them are defined based on the general schema: when P does an action of interest (passes a test, produces a barb, etc) then Q should be able to match it, employing an existential quantifier. Moreover, we would like to investigate models in which both angelic and demonic non-determinism are present. One approach would be to use two separate schedulers, one acting in favour and one against the process, along the lines of [21].

References

1. Roscoe, A.W.: Modelling and verifying key-exchange protocols using CSP and FDR. In: Proc. CSFW, IEEE Computer Soc Press (1995) 98–107

2. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Information and Computation **148** (1999) 1–70
3. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: 28th Annual Symposium on Principles of Programming Languages (POPL), ACM (2001) 104–115
4. Kremer, S., Ryan, M.D.: Analysis of an electronic voting protocol in the applied pi-calculus. In: Proc. of ESOP'05. Volume 3444 of LNCS., Springer (2005) 186–200
5. McLean: A general theory of composition for a class of "possibilistic" properties. IEEETSE: IEEE Transactions on Software Engineering **22** (1996)
6. Roscoe, B.: CSP and determinism in security modelling. In: Proc. of 1995 IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1995)
7. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In Abadi, M., de Alfaro, L., eds.: Proceedings of CONCUR. Volume 3653 of LNCS., Springer (2005) 171–185
8. Chatzikokolakis, K., Palamidessi, C.: Making random choices invisible to the scheduler. In: Proc. of CONCUR'07. Volume 4703 of LNCS., Springer (2007) 42–58
9. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, MIT (1995)
10. Chatzikokolakis, K.: Probabilistic and Information-Theoretic Approaches to Anonymity. PhD thesis, Ecole Polytechnique, Paris (2007)
11. Baier, C.: Polynomial-time algorithms for testing probabilistic bisimulation and simulation. LNCS **1102** (1996) 50–61
12. DemonicChecker: A tool to verify demonic bisimulation for finite processes (2008) http://www.win.tue.nl/ kostas/demonicchecker/.
13. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology **1** (1988) 65–75
14. Canetti, R., Cheung, L., Kaynar, D., Liskov, M., Lynch, N., Pereira, O., Segala, R.: Task-structured probabilistic i/o automata. In: Proceedings the 8th International Workshop on Discrete Event Systems (WODES'06), Ann Arbor, Michigan (2006)
15. Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Time-bounded task-PIOAs: A framework for analyzing security protocols. In Dolev, S., ed.: Proceedings of the 20th International Symposium in Distributed Computing (DISC '06). Volume 4167 of LNCS., Springer (2006) 238–253
16. Garcia, F.D., van Rossum, P., Sokolova, A.: Probabilistic anonymity and admissible schedulers (2007) arXiv:0706.1019v1.
17. Jurjens: Secrecy-preserving refinement. In: International Symposium on FME 2001: Formal Methods for Increasing Software Productivity, LNCS. Volume 10. (2001)
18. Mantel, H.: Possibilistic definitions of security - an assembly kit. In: CSFW. (2000) 185–199
19. Lincoln, P., Mitchell, J., Mitchell, M., Scedrov, A.: A probabilistic poly-time framework for protocol analysis. In: Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, California, ACM Press (1998) 112–121
20. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: Proc. of the European Symposium on Research in Computer Security (ESORICS). Volume 1146 of LNCS., Springer (1996) 198–218
21. Chatzikokolakis, K., Knight, S., Panangaden, P.: Epistemic strategies and games on concurrent processes. In: Proc. of SOFSEM'08. (2008) To appear.

A Proofs

We give here the proofs of the main results of the paper:

Definition 7. *Let A, B be two countable sets, μ be a probability discrete measure on A and f be a function from A to B. We define $f(\mu)$ as the discrete probability measure on B such that*

$$f(\mu)(\{b\}) = \mu(f^{-1}b)$$

Lemma 1. *Let A, B be two countable sets, $\sim_A, \sim_B$ be two equivalence relations on A, B respectively and f be a function from A to B such that $a_1 \sim_A a_2 \Rightarrow fa_1 \sim_B fa_2, \forall a_1, a_2 \in A$. Also, let μ, κ be two discreate probability measures on A. Then $\mu \sim_A \kappa \Rightarrow f(\mu) \sim_B f(\kappa)$.*

Proof. Let $[b]_{\sim_B}, b \in B$ be an equivalence class of $\sim_B$. If $a \in f^{-1}([b]_{\sim_B})$ then $a' \in f^{-1}([b]_{\sim_B})$ for all $a' \sim_A a$. Thus, for all equivalent classes $C \in A/\sim_A$ we have either $C \subseteq f^{-1}([b]_{\sim_B})$ or $C \cap f^{-1}([b]_{\sim_B}) = \emptyset$. Let

$$C_b = \{C \in A/\sim_A \mid C \subseteq f^{-1}([b]_{\sim_B})\}$$

we have

$$f(\mu)([b]_{\sim_B}) = \mu(f^{-1}[b]_{\sim_B}) = \mu(\cup_{C \in C_b} C) = \textstyle\sum_{C \in C_b} \mu(C)$$

Thus, if $\mu \sim_A \kappa$ then for all $b \in B$:

$$f(\mu)([b]_{\sim_B}) = \textstyle\sum_{C \in C_b} \mu(C) = \sum_{C \in C_b} \kappa(C) = f(\kappa)([b]_{\sim_B})$$

which means that $f(\mu) \sim_B f(\kappa)$.

Lemma 2. *Let A, B be two countable sets, $\sim_A, \sim_B$ be two equivalence relations on A, B respectively and $\{f_i | i \in \mathrm{N}\}$ be a set of functions from A to B such that*

$$\forall a_1, a_2 \in A : (a_1 \sim_A a_2 \;\Leftrightarrow\; \forall i \in \mathrm{N} : f_i a_1 \sim_B f_i a_2)$$

Also, let μ, κ be two discreate probability measures on A. Then $\mu \sim_A \kappa \Leftrightarrow \forall i \in \mathrm{N} : f_i(\mu) \sim_B f_i(\kappa)$.

Proof.

Lemma 3. *Let $P \in \mathcal{P}$ be a process with a deterministic labeling and S a finite scheduler other than 0. Then*

$$\begin{aligned} & P \parallel S \xrightarrow{\alpha}_s \mu \parallel S' \\ \textit{iff} \quad & \varphi_S(P) \xrightarrow{\alpha}_c \varphi_{S'}(\mu) \end{aligned}$$

Proof. By Induction on (P, S), using a lexicographic combination of the sub-process and sub-scheduler order. Cases (i)-(vi) assume that the scheduler is not an **if-then-else**, the last case treats such schedulers. The two directions are similar so we treat them together.
Base cases

i) $P = l : \alpha.P'$: The scheduler must be of the form $S = l.S'$. Then $P \parallel S$ and $\varphi_S(P) = l\!:\!\alpha.\varphi_{S'}(P')$ can both make a transition α to $\delta(P') \parallel S$ and $\delta(\varphi_{S'}(P'))$ respectively.

ii) $P = l : \sum_i p_i P_i$: The scheduler must be of the form $S = l.S'$ and $\varphi_S(P) = l\!:\!\sum_i p_i \varphi_{S'}(P_i)$. Then

$$\begin{aligned} &P \parallel S \xrightarrow{\tau}_s \mu \parallel S' && \mu = \textstyle\sum_i [p_i]\delta(P_i) \\ &\varphi_S(P) \xrightarrow{\tau}_c \mu' && \mu' = \textstyle\sum_i [p_i]\delta(\varphi_{S'}(P_i)) \end{aligned}$$

and $\mu'(P') = \mu(\varphi_{S'}^{-1}(P'))\ \forall P'$ follows.

iii) $P = !l : a.P'$: similar to case (i)

Inductive cases:

iv) $P = P_1 + P_2$: Assuming that the transition $P \parallel S \xrightarrow{\alpha}_s \mu \parallel S'$ is generated by the SUM1 rule, then $P_1 \parallel S$ should be able to perform the same transition. By the induction hypothesis we have $\varphi_S(P_1) \xrightarrow{\alpha}_c \mu'$ with $\mu'(P') = \mu(\varphi_{S'}^{-1}(P'))\ \forall P'$. Then, $\varphi_S(P_1) + \varphi_S(P_2)$ can do the same transition using the SUM1 rule. The case of SUM2 as well as the other direction are similar.

v) $P = (\nu a)P'$: similar to case (iv)

vi) $P = P_1 \mid P_2$: A transition of P is generated either by PAR1, PAR2 or COM rule. Consider first PAR1 with $S = \lambda.S'$ and $P_1 \parallel \lambda.S' \xrightarrow{\alpha}_s \delta(P_1') \parallel S'$. Then, by induction hypothesis, $\varphi_S(P_1) \xrightarrow{\alpha}_c \delta(P_1')$ and by definition of φ: $\varphi_S(P_1 \mid P_2) = \lambda : \alpha.\varphi_{S'}(P_1' \mid P_2)$. Finally, the last process can make a transition α to $\delta(\varphi_{S'}(P_1' \mid P2))$.
All the other cases (non-δ transition, PAR2, COM) are similar, as well as the inverse direction.

vii) $S = \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2$. If $P \parallel S \xrightarrow{\alpha}_s \mu \parallel S'$ by the IF1 rule, then $l \in tp(P)$ and $P_1 \parallel S \xrightarrow{\alpha}_s \mu \parallel S'$. Then by the induction hypothesis for (P, S_1) we have $\varphi_{S_1}(P) \xrightarrow{\alpha}_c \mu'$ with $\mu'(P') = \mu(\varphi_{S'}^{-1}(P'))\ \forall P'$. Finally by the definition of φ we have $\varphi_S(P) \xrightarrow{\alpha}_c \mu'$. Similarly for IF2.

Theorem 1 The equivalence relations $\sim_R$ and $\sim_D$ coincide.

Proof. $\Rightarrow$) We show that $\sim_D$ is an R-bisimulation. We need to show that

$$P \sim_D Q \;\Rightarrow \varphi_S(P) \sim \varphi_S(Q) \quad \forall P, Q \tag{13}$$

for all finite schedulers S. The proof is by induction on the structure of S.
Base case $S = 0$:

$$\varphi_0(P) = P \sim Q = \varphi_0(Q)$$

since $\sim_D \subseteq \sim$.
Inductive case: Let $(P, Q) \in \sim_D$ and let S be a finite scheduler. We have:

$$\begin{aligned}
\varphi_S(P) &\xrightarrow{\alpha}_c \mu' \Rightarrow & \\
P \parallel S &\xrightarrow{\alpha}_s \mu \parallel S' \Rightarrow & \text{lemma 3} \\
Q \parallel S &\xrightarrow{\alpha}_s \kappa \parallel S' \Rightarrow & P \sim_D Q \\
\varphi_S(Q) &\xrightarrow{\alpha}_c \kappa' & \text{lemma 3}
\end{aligned}$$

with

$$\begin{gathered}
\mu'(P') = \mu(\varphi_{S'}^{-1}(P') \ \forall P' \in \mathcal{P} \\
\mu \sim_D \kappa \\
\kappa'(Q') = \kappa(\varphi_{S'}^{-1}(Q') \ \forall Q' \in \mathcal{P}
\end{gathered}$$

Then, from the induction hypothesis for S' (13) we get that the function $\varphi_{S'}$: $\mathcal{P} \to \mathcal{P}$ satisfies the conditions of lemma 1 where $A = B = \mathcal{P}$, $\sim_A = \sim_D$ and $\sim_B = \sim$. From this we get $\mu' \sim \kappa$, which implies that $\varphi_S(P) \sim \varphi_S(Q)$

$\Leftarrow$) We show that $\sim_R$ is a demonic bisimulation. We need to show that

$$P \parallel S \xrightarrow{\alpha}_s \mu \parallel S' \ \Rightarrow \ Q \parallel S \xrightarrow{\alpha}_s \kappa \parallel S' \quad \text{with } \mu \sim_R \kappa \qquad \forall (P, Q) \in \sim_R$$

$(P, Q) \in \sim_R$ and let S be a non-blocking scheduler for P. Then $P \sim$

Theorem 2 The equivalence relations $\sim_D, \sim_A$ coincide.

Proof. First we notice that the rules of the semantics for complete processes, except from IF1,IF2, closely match those of the semantics without schedulers. Thus it is easy to see that

$$P \parallel l.S \xrightarrow{\alpha} \mu \parallel S \quad \Leftrightarrow \quad P \xrightarrow{l:\alpha}_a \mu \tag{14}$$

$$P \parallel (l_1, l_2).S \xrightarrow{\alpha} \mu \parallel S \quad \Leftrightarrow \quad P \xrightarrow{(l_1,l_2):\alpha}_a \mu \tag{15}$$

$\Leftarrow$) We first show that $\sim_A$ is a demonic bisimulation. Let $P_1 \sim_A$ and let S be a non-blocking scheduler for P_1 such that

$$P_1 \parallel S \xrightarrow{\alpha} \mu_1 \parallel S' \tag{16}$$

We perform a case analysis on S:

i) $S = l.S'$. From (14),(16) we have that $P_1 \xrightarrow{l:\alpha}_a \mu_1$. Since $P_1 \sim_A P_2$ we have $P_2 \xrightarrow{l:\alpha}_a \mu_2$ with $\mu_1 \sim_A \mu_2$. And again from (14): $P_2 \parallel S \xrightarrow{\alpha} \mu_2 \parallel S'$.
ii) $S = (l_1, l_2).S'$. Similar to (i), using (15).

iii) S starts with an arbitrary number of nested **if**s. In this case only the IF1,IF2 rules are applicable and in the premises of the first IF rule applied we will have a transition of the form

$$P_1 \parallel S_l \xrightarrow{\alpha} \mu_1 \parallel S'$$

where S_l is a scheduler that does not start with an **if**. Now the cases (i),(ii) apply, thus a transition $P_2 \parallel S_l \xrightarrow{\alpha} \mu_2 \parallel S'$ with $\mu_1 \sim_A \mu_2$ will be enabled. Finally, since $P_1 \sim_A P_2$ we have $tl(P_1) = tl(P_2)$. Hence, the IF rules can be applied in the same way for P_2 giving a transition $P_2 \parallel S \xrightarrow{\alpha} \mu_2 \parallel S'$.

This shows that $\sim_A$ is a demonic bisimulation, which means that $\sim_D \supseteq \sim_A$.
$\Rightarrow$) We now show that $\sim_D$ is an A-bisimulation. Let $P_1 \sim_D P_2$ and $P_1 \xrightarrow{l:\alpha} \mu_1$.

i) From (15) we have that $P_1 \parallel l.S \xrightarrow{\alpha} \mu_1 \parallel S$. Since $P_1 \sim P_2$ we have $P_2 \parallel l.S \xrightarrow{\alpha} \mu_2 \parallel S$ with $\mu_1 \sim_D \mu_2$ which in turn gives $P_2 \xrightarrow{l:\alpha} \mu_2$.
ii) We need to show that $tl(P_1) = tl(P_2)$. Assuming otherwise, let $l' \in tl(P_1) \setminus tl(P_2)$. Then the scheduler **if** l' **then** l **else** 0 produces a transition for P_1 but not for P_2 which violates the assumption $P_1 \sim_D P_2$.

Similarly for the case $P_1 \xrightarrow{(l_1,l_2):\alpha} \mu_1$. This shows that $\sim_D$ is an A-bisimulation, which means that
$\sim_D \subseteq \sim_A$, so finally we have $\sim_D = \sim_A$. □

Proposition 3 If $Prot_i \sim_D Prot_j$ for all i, j then the protocol satisfies strong probabilistic anonymity (Def. 5)

Proof. Follows from the fact that the fully probabilistic automata $Prot_i \parallel S$ and $Prot_j \parallel S$ will be probabilistically bisimilar, and that probabilistic bisimulation implies the same distribution of traces.

Proposition 4. *For any number of cryptographers, $Prot_i \sim_D Prot_j$, where $Prot_i, Prot_j$ are the processes defined in Figure 6.*

Proof. (sketch) Define

$$\begin{aligned} Chain_{i,j} &\triangleq Crypt_i \mid Coin_i \mid \ \dots \ \mid Coin_{j-1} \mid Crypt_j \\ \mathbf{0}_i &\triangleq \bar{c}_i\langle 0 \rangle \\ \mathbf{1}_i &\triangleq \bar{c}_i\langle 1 \rangle \end{aligned}$$

So $\mathbf{0}_i$ is the process that sends 0 to cryptographer i, and $Coin_i = (\mathbf{0}_i \mid \mathbf{0}_{i\oplus 1}) +_{0.5} (\mathbf{1}_i \mid \mathbf{1}_{i\oplus 1})$. First, we show that

$$\mathbf{0}_i \mid Crypt_i \mid \mathbf{0}_i \sim_D \mathbf{1}_i \mid Crypt_i \mid \mathbf{1}_i \qquad (17)$$

which is expected, since cryptographer i outputs the xor of the two coins. This equivalence could be also verified automatically. Then, by induction on $j-i$, we can show that

$$\mathbf{0}_i \mid Chain_{i,j} \mid \mathbf{0}_j \sim_D \mathbf{1}_i \mid Chain_{i,j} \mid \mathbf{1}_j \tag{18}$$
$$\mathbf{0}_i \mid Chain_{i,j} \mid \mathbf{1}_j \sim_D \mathbf{1}_i \mid Chain_{i,j} \mid \mathbf{0}_j \tag{19}$$

The base case, $i = j$, is exactly (17). For the inductive case we have for (18):

$$\mathbf{0}_i \mid Crypt_i \mid Coin_i \mid Chain_{i+1,j} \mid \mathbf{0}_j \overset{?}{\sim}_D \mathbf{1}_i \mid Crypt_i \mid Coin_i \mid Chain_{i+1,j} \mid \mathbf{1}_j \tag{20}$$

To establish this equivalence we look at the transitions of each side. The more interesting case is a transition of $Coin_i$. With probability 1/2 the left-hand side can perform a τ transition to

$$\begin{aligned}&\mathbf{0}_i \mid Crypt_i \mid \mathbf{0}_i \mid \mathbf{0}_{i+1} \mid Chain_{i+1,j} \mid \mathbf{0}_j \sim_D\\ &\mathbf{1}_i \mid Crypt_i \mid \mathbf{1}_i \mid \mathbf{1}_{i+1} \mid Chain_{i+1,j} \mid \mathbf{1}_j \qquad \text{by (17) and ind. hypothesis}\end{aligned}$$

which can be reached from the right-hand side of (20) by a τ transition with probability 1/2. Similarly for the case that $Coin_i$ is resolved to 1. Thus, transitions go with the same probability to bisimilar processes, so we can show (20) by constructing a proper bisimulation relation, which completes the proof of (18). The case of (19) is identical.

Now we want to compare two instances of the protocol with different payers. Assume, for simplicity that the payers are adjacent, say users 1 and 2. Then the two instances to compare are

$$\begin{aligned}&Prot_1 = Coin_0 \mid CryptP_1 \mid Coin_1 \mid Crypt_2 \mid Coin_2 \mid Chain_{3,n} \text{ and}\\ &Prot_2 = Coin_0 \mid Crypt_1 \mid Coin_1 \mid CryptP_2 \mid Coin_2 \mid Chain_{3,n}\end{aligned}$$

We now show that

$$\begin{aligned}&\mathbf{0}_1 \mid CryptP_1 \mid Coin_1 \mid Crypt_2 \mid \mathbf{0}_2 \sim_D \mathbf{1}_1 \mid Crypt_1 \mid Coin_1 \mid CryptP_2 \mid \mathbf{1}_2\\ &\mathbf{0}_1 \mid CryptP_1 \mid Coin_1 \mid Crypt_2 \mid \mathbf{1}_2 \sim_D \mathbf{1}_1 \mid Crypt_1 \mid Coin_1 \mid CryptP_2 \mid \mathbf{0}_2\end{aligned}$$

looking at the transitions of $Coin_1$ and using the same argument as for (18),(19). We then proceed with

$$\begin{aligned}&\mathbf{0}_1 \mid CryptP_1 \mid Coin_1 \mid Crypt_2 \mid Coin_2 \mid Chain_{3,n} \mid \mathbf{0}_n \sim_D\\ &\mathbf{1}_1 \mid Crypt_1 \mid Coin_1 \mid CryptP_2 \mid Coin_2 \mid Chain_{3,n} \mid \mathbf{1}_n \qquad \text{and}\\ &\mathbf{0}_1 \mid CryptP_1 \mid Coin_1 \mid Crypt_2 \mid Coin_2 \mid Chain_{3,n} \mid \mathbf{1}_n \sim_D\\ &\mathbf{1}_1 \mid Crypt_1 \mid Coin_1 \mid CryptP_2 \mid Coin_2 \mid Chain_{3,n} \mid \mathbf{0}_n\end{aligned}$$

by examining the transitions of $Coin_2$, and finally we do a case analysis on $Coin_0$ and show that

$$Prot_1 \sim_D Prot_2$$

The advantage of working with an equivalence here is that we can show a property of arbitrarily long chains of cryptographers ((18),(19)) and use this property in bigger contexts. Note that, all the above equivalences hold only if the "coin channels" c_i in the corresponding processes are restricted (we omitted the restriction to simplify the notation). Then, when we compose them to form $Prot_1, Prot_2$ we have to move these restrictions to the outside of $Prot_1, Prot_2$. This is however possible, since in all these processes (eg. those of (17)) the coin channels appearing are not used by any other subprocesses of the protocol.

The case where the payers are not adjacent can be treated in a similar way, this time using two chains of cryptographers.

$$
\begin{aligned}
Prot_1 &= Coin_0 \mid CryptP_1 \mid Coin_1 \mid Chain_{1,i-1} \mid Coin_{i-1} \mid Crypt_i \mid Coin_i \mid Chain_{i+1,n} \\
Prot_i &= Coin_0 \mid Crypt_1 \mid Coin_1 \mid Chain_{1,i-1} \mid Coin_{i-1} \mid CryptP_i \mid Coin_i \mid Chain_{i+1,n}
\end{aligned}
$$

$\square$