# Safe Equivalences for Security Properties

Mário S. Alvim[1], Miguel E. Andrés[2], Catuscia Palamidessi[1], and Peter van Rossum[2].

[1]INRIA and LIX, École Polytechnique Palaiseau, France.
[2]Institute for Computing and Information Sciences, The Netherlands.

**Abstract.** In the field of Security, process equivalences have been used to characterize various information-hiding properties (for instance secrecy, anonymity and non-interference) based on the principle that a protocol $P$ with a variable $x$ satisfies such property if and only if, for every pair of secrets $s_1$ and $s_2$, $P[s_1/x]$ is equivalent to $P[s_2/x]$. We argue that, in the presence of nondeterminism, the above principle relies on the assumption that the scheduler "works for the benefit of the protocol", and this is usually not a safe assumption. Non-safe equivalences, in this sense, include (partial-) trace equivalence, bisimulation and testing. We present a formalism in which we can specify admissible schedulers and, correspondingly, safe versions of these equivalences. We then show these variants are still congruences. Finally, we investigate the relation with the recent notion of "demonic bisimulation" proposed by Chatzikokolakis, Gethin and Parker.

## 1 Introduction

Let $\widetilde{a} \in \Sigma^\star$, we define the set of paths with trace $\widetilde{a}$ as $[\widetilde{a}] \stackrel{\text{def}}{=} \{\sigma \in \text{CPaths}(M) \mid trace(\sigma) = \widetilde{a}\}$.

*Motivation* Consider the system

$$S \stackrel{\text{def}}{=} (c, out)(A \,||\, H_1 \,||\, H_2 \,||\, Cor\,)$$

where

$$A \stackrel{\text{def}}{=} \overline{c}(sec) \quad H_1 \stackrel{\text{def}}{=} c(s).\overline{out}\langle a\rangle \quad H_2 \stackrel{\text{def}}{=} c(s).\overline{out}\langle b\rangle \quad Corr \stackrel{\text{def}}{=} c(s).\overline{out}\langle s\rangle$$

Then we have $S\left\{{}^{a}/_{sec}\right\} \equiv S\left\{{}^{b}/_{sec}\right\}$.

Schedulers:

**Bad:** They can see the secrets and induce a leakage by making different choices based on the different secrets

**Neutral (safe):** They do not see the secrets. Same choices for different secrets.

**Good:** They see the secrets and protect from leakage by making different choices based on the secrets (this is what the "bad" use of bisimulation is based on)

| | Demonic | Angelic |
|---|---|---|
| **Restricted** | | |
| **Omniscient** | | |

**Table 1.**

## 2 Preliminaries

In this section we gather preliminary notions and results related to probabilistic automata [**?**,**?**].

### 2.1 Probabilistic automata

**Mig:** The support of $\mu$ does not need to be countable anymore

**Cat:** In the non countable case we would need to use an integral, so let's leave it this way

A function $\mu\colon Q \to [0,1]$ is a *discrete probability distribution* on a set $Q$ if the support of $\mu$ is countable and $\sum_{q\in Q} \mu(q) = 1$. The set of all discrete probability distributions on $Q$ is denoted by $\mathcal{D}(Q)$.

A *probabilistic automaton* is a quadruple $M = (Q, \Sigma, \hat{q}, \alpha)$ where

- $Q$ is a countable set of *states*,
- $\Sigma$ a finite set of *actions*,
- $\hat{q}$ the *initial* state, and
- $\alpha$ a *transition function* $\alpha : Q \to \mathcal{P}(\Sigma \times \mathcal{D}(Q))$.

Where $\mathcal{P}(X)$ is the set of all finite subsets of $X$.

If $\alpha(q) = \emptyset$ then $q$ is a *terminal* state. We write $q \xrightarrow{a} \mu$ for $(a,\mu) \in \alpha(q)$. Moreover, we write $q \xrightarrow{a} r$ whenever $q \xrightarrow{a} \mu$ and $\mu(r) > 0$. A *fully probabilistic automaton* is a probabilistic automaton satisfying $|\alpha(q)| \leq 1$ for all states. In case $\alpha(q) \neq \emptyset$ in a fully probabilistic automaton, we will overload notation and use $\alpha(q)$ to denote the distribution outgoing from $q$.

**Cat:** Should we say that these are called "simple Probabilistic Automata", althoug we will often use "Probabilistic Automata" for simplicity? In the rest of the paper I have used "simple" to point out the difference with the QEST paper.

A *path* in a probabilistic automaton is a sequence $\sigma = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots$ where $q_i \in Q$, $a_i \in \Sigma$ and $q_i \xrightarrow{a_{i+1}} q_{i+1}$. A path can be *finite* in which case it ends with a state. A path is *complete* if it is either infinite or finite ending in a terminal state. Given a path $\sigma$, $first(\sigma)$ denotes its first state, and if $\sigma$ is finite then $last(\sigma)$ denotes its last state. Let $\mathrm{Paths}_q(M)$ denote the set of all paths, $\mathrm{Paths}^\star_q(M)$ the set of all finite paths, and $\mathrm{CPaths}_q(M)$ the set of all complete paths of an automaton $M$, starting from the state $q$. We will omit $q$ if $q = \hat{q}$. Paths are ordered by the prefix relation, which we denote by $\leq$. The *trace* of a path is the sequence of actions in $\Sigma^* \cup \Sigma^\infty$ obtained by removing the states, hence for the above path $\sigma$ we have $trace(\sigma) = a_1 a_2 \ldots$. If $\Sigma' \subseteq \Sigma$, then $trace_{\Sigma'}(\sigma)$ is the projection of $trace(\sigma)$ on the elements of $\Sigma'$. The *length* of a finite path $\sigma$, denoted by $|\sigma|$, is the number of actions in its trace.

**Mig:** Catuscia: Is this what you like for first, last, etc?

**Cat:** Yes, thanks

Let $M = (Q, \Sigma, \hat{q}, \alpha)$ be a (fully) probabilistic automaton, $q \in Q$ a state, and let $\sigma \in \mathrm{Paths}^\star_q(M)$ be a finite path starting in $q$. The *cone* generated by $\sigma$ is the set of complete paths $\langle\sigma\rangle = \{\sigma' \in \mathrm{CPaths}_q(M) \mid \sigma \leq \sigma'\}$. Given a fully probabilistic automaton $M = (Q, \Sigma, \hat{q}, \alpha)$ and a state $q$, we can calculate the *probability value*, denoted by $\mathbf{P}_q(\sigma)$, of any finite path $\sigma$ starting in $q$ as follows: $\mathbf{P}_q(q) = 1$ and $\mathbf{P}_q(\sigma \xrightarrow{a} q') = \mathbf{P}_q(\sigma) \cdot \mu(q'),$ where $last(\sigma) \xrightarrow{a} \mu$.

Let $\Omega_q \stackrel{\text{def}}{=} \mathrm{CPaths}_q(M)$ be the sample space, and let $\mathcal{F}_q$ be the smallest $\sigma$-algebra generated by the cones. Then $\mathbf{P}_q$ induces a unique *probability measure* on $\mathcal{F}_q$ (which we will also denote by $\mathbf{P}_q$) such that $\mathbf{P}_q(\langle\sigma\rangle) = \mathbf{P}_q(\sigma)$ for every finite path $\sigma$ starting in $q$. For $q = \hat{q}$ we write $\mathbf{P}$ instead of $\mathbf{P}_{\hat{q}}$.

A scheduler for a probabilistic automaton $M$ is a function

$$\zeta \colon \mathrm{Paths}^\star(M) \to (\Sigma \times \mathcal{D}(Q) \cup \bot)$$

satisfying $\zeta(\sigma) = (a, \mu)$ implies $last(\sigma) \xrightarrow{a} \mu$, for each finite path $\sigma$.
Hence, a scheduler selects an available transitions in each state. It is history dependent since it takes into account the path (history) and not only the current state. It is partial since it gives a sub-probability distribution, i.e., it may halt the execution at any time.

## 3 Systems

In this section we describe the kind of systems we are dealing with. We start by introducing a variant of probabilistic automata, that we call *Tagged Probabilistic Automata*. These systems are parallel compositions of probabilistic processes, called *components*. Each component is equipped with a unique identifier, called *tag*. Whenever a component (or a pair of components in case of synchronization) makes a step, the corresponding transition will be decorated with the associated tag (or pair of tags).

**Cat:** I have removed the requirement that the components are purely probabilistic. It's not needed here

Similar systems have been already introduced in [**?**]. The main difference is that here the components are simple probablistic automata, i.e. they may contain internal nondetermism, and each transition goes from a node and a label to a distributions over nodes. In [**?**] the components are fully probabilistic (except for the input guards, that may receive different values), and the secrets can appear only in a probabilistic choice.

**Cat:** Make sure we introduce simple (labeled) probabilistic automata in the preliminaries. In this paper, we will use the word "label" to refer to pairs tag: action or (tag,tag):action.

**Definition 1.** *A* Tagged Probabilistic Automaton *(TPA) is a tuple* $(Q, L, \Sigma, \hat{q}, \alpha)$*, where*

- $Q$ *is a set of* states,
- $L$ *is a set of* tags,
- $\Sigma$ *is a set of* actions,
- $\hat{q} \in Q$ *is the* initial state,
- $\alpha \colon Q \to \mathcal{P}(L \times \Sigma \times \mathcal{D}(Q))$ *is a* transition function.

**Cat:** I have eliminated the restriction that $\alpha$ should be a finite set of transitions

In the following we write $q \xrightarrow{l:a} \mu$ for $(\ell, a, \mu) \in \alpha(q)$, and we use $Enabled(q)$ to denote the tags of the components that are enabled to make a transition. Namely,

$$Enabled(q) \stackrel{\text{def}}{=} \{\ell \in L \mid there\ exists\ a \in \Sigma, \mu \in \mathcal{D}(Q)\ such\ that\ q \xrightarrow{l:a} \mu\}$$

In these systems, we can decompose the scheduler in two: a *global scheduler*, which decides which component or pair of components makes the move next, and a *local scheduler*, which solves the internal nondeterminism of the selected component.

We assume that the local scheduler can select only a transition which is enabled, and that the global scheduler can only select a component among those which are enabled. This means that the execution does not stop unless all components are blocked (suspended or terminated). This is in line with the spirit of process algebra, and also with

the tradition of Markov Decision Processes, but contrasts with that of the Probabilistic Automata of Lynch and Segala [**?**]. However, the results in this paper do not depend on this assumption; we could as well allow schedulers which decide to terminate the execution even though there are transitions enabled in the last state.

**Definition 2.** *Let $M = (Q, L, \Sigma, \hat{q}, \alpha)$ be a Tagged Probabilistic Automaton.*

- *A global scheduler for $M$ is a function $\zeta \colon \mathrm{Paths}^\star(M) \to (L \cup \{\bot\})$ such that for all finite paths $\sigma$, if $Enabled(last(\sigma)) \neq \emptyset$ then $\zeta(\sigma) \in Enabled(last(\sigma))$, and $\zeta(\sigma) = \bot$ otherwise.*
- *A local scheduler for $M$ is a function $\xi \colon \mathrm{Paths}^\star(M) \to (L \times \Sigma \times \mathcal{D}(Q) \cup \{\bot\})$ such that, for all finite paths $\sigma$, if $\alpha(last(\sigma)) \neq \emptyset$ then $\xi(\sigma) \in \alpha(last(\sigma))$, and $\xi(\sigma) = \bot$ otherwise.*
- *A global scheduler $\zeta$ and a local scheduler $\xi$ for $M$ are* compatible *if, for all finite paths $\sigma$, $\xi(\sigma) = (\ell, a, \mu)$ implies $\zeta(\sigma) = \ell$, and $\xi(\sigma) = \bot$ implies $\zeta(\sigma) = \bot$.*
- *A scheduler for $M$ is a pair $(\zeta, \xi)$ of compatible global and local schedulers for $M$.*

**Cat:** Please use italics for "last", and for all the other function names.

We are going to use a simple probabilistic process calculus (a sort of probabilistic version of CCS [**?**,**?**]) to specify the components.

*Components' syntax:* We assume a set of *actions* $\Sigma$ with elements $a, a_1, a_2, \cdots$, including the special symbol $\tau$ denoting a *silent step*. With the exception of $\tau$, each action $a$ has a unique co-action $\bar{a} \in \Sigma$ and we assume $\bar{\bar{a}} = a$.

A component $q$ is a process specified by the following grammar:

| | | | |
|---|---|---|---|
| **Components** | $q$ ::= | $0$ | termination |
| | | $a.q$ | prefix |
| | | $q_1 + q_2$ | nondeterministic choice |
| | | $\sum_i p_i : q_i$ | probabilistic choice |
| | | $q_1 \mid q_2$ | parallel composition |
| | | $(a)q$ | restriction |
| | | $A$ | process call |

The $p_i$, in the blind and secret choices, represents the probability of the $i$-th branch and must satisfy $0 \leq p_i \leq 1$ and $\sum_i p_i = 1$. The process call $A$ is a simple process identifier. For each identifier, we assume a corresponding unique process declaration of the form $A \stackrel{\mathrm{def}}{=} q$. The idea is that, whenever $A$ is executed, it triggers the execution of $q$. Note that $q$ can contain $A$ or another process identifier, which means that our language allows (mutual) recursion.

*Components' semantics:* The operational semantics consists of probabilistic transitions of the form $q \stackrel{a}{\rightarrow} \mu$ where $q \in Q$ is a process, $a \in \Sigma$ is an action and $\mu \in \mathcal{D}(Q)$ is a distribution on processes. They are specified by the following rules:

$$\text{PRF}\ \frac{}{a.q \xrightarrow{a} \delta_q} \qquad \text{NDT}\ \frac{q_1 \xrightarrow{a} \mu}{q_1 + q_2 \xrightarrow{a} \mu}$$

$$\text{PRB}\ \frac{}{\sum_i p_i : q_i \xrightarrow{\tau} \sum\!\!\!\!\!\!\sum_i p_i \cdot \delta_{q_i}} \qquad \text{PAR}\ \frac{q_1 \xrightarrow{a} \mu}{q_1 \mid q_2 \xrightarrow{a} \mu \mid q_2}$$

$$\text{CALL}\ \frac{q \xrightarrow{a} \mu}{A \xrightarrow{a} \mu}\ \text{if } A \stackrel{\text{def}}{=} q \qquad \text{COM}\ \frac{q_1 \xrightarrow{a} \delta_{r_1} \quad q_2 \xrightarrow{\bar{a}} \delta_{r_2}}{q_1 \mid q_2 \xrightarrow{\tau} \delta_{r_1 | r_2}}$$

$$\text{RST}\ \frac{q \xrightarrow{a} \mu}{(b)q \xrightarrow{a} \mu}\ a, \bar{a} \neq b$$

We assume also the symmetric versions of the rules NDT, PAR and COM. The symbol $\delta_q$ is the delta of Dirac, which assigns probability 1 to $q$ and 0 to all other processes. The symbol $\sum\!\!\!\!\!\!\sum_i$ represents summation on distributions. Namely, $\sum\!\!\!\!\!\!\sum_i p_i \cdot \mu_i$ is the distribution $\mu$ such that $\mu(x) = \sum_i p_i \cdot \mu_i(x)$. The notation $\mu \mid q$ represents the distribution $\mu'$ such that $\mu'(r) = \mu(q')$ if $r = q' \mid q$, and $\mu'(r) = 0$ otherwise.

*Systems* A system is composed by $n$ processes (components) in parallel, and restricted at the top-level on a subset of actions $A \subseteq \Sigma$:

$$(A)\ q_1 \parallel q_2 \parallel \cdots \parallel q_n.$$

The restriction on $A$ enforces synchronization on the channel names belonging to $A$, in accordance with the CCS spirit.

*Systems' semantics* The semantics of a system gives rise to a TPA, where the states are terms representing systems during their evolution. A transition now is of the form $q \xrightarrow{\ell:a} \mu$ where $a \in \Sigma$, $\mu \in \mathcal{D}(Q)$, and $\ell \in L$ is either the tag of the component which makes the move, or a (unordered) pair of tags representing the two partners of a synchronization. We will set $L$ to be the indexes of the componets, i.e. $L = I \cup I^2$ where $I = \{1, 2, \ldots, n\}$.

*Interleaving*

$$\frac{q_i \xrightarrow{a} \sum\!\!\!\!\!\!\sum_j p_j \cdot \delta_{q_{ij}}}{(A)\ q_1 \parallel \cdots \parallel q_i \parallel \cdots \parallel q_n \xrightarrow{i:a} \sum\!\!\!\!\!\!\sum_j p_j \cdot \delta_{(A)q_1 \parallel \cdots \parallel q_{ij} \parallel \cdots \parallel q_n}} \qquad a \notin A$$

where $i$ is the tag indicating that the component $i$ is making the step.

Note that we assume that the probabilistic choices in the syntax of the components are finite. This implies that every transition $q \xrightarrow{\ell:a} \mu$ can be written as $q \xrightarrow{\ell:a} \sum\!\!\!\!\!\!\sum_i p_i \cdot \delta_{q_i}$, thus justifying the notation used in the interleaving rule.

*Synchronization*

$$\frac{q_i \xrightarrow{a} \delta_{q_i'} \qquad q_j \xrightarrow{\bar{a}} \delta_{q_j'}}{(A)\ q_1 \parallel \cdots \parallel q_i \parallel \cdots \parallel q_j \parallel \cdots \parallel q_n \xrightarrow{i,j:\tau} \delta_{(A)q_1\|\cdots\|q_i'\|\cdots\|q_j'\|\cdots\|q_n}}$$

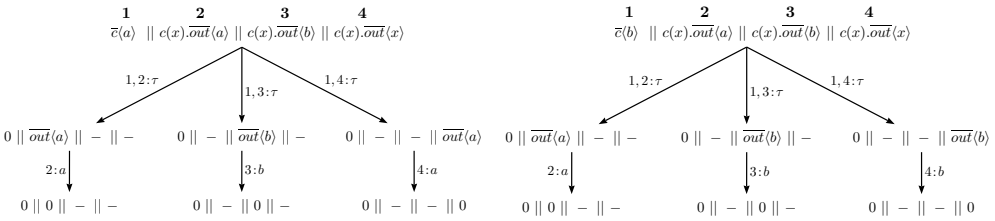
here $\{i, j\}$ is the tag indicating that the components making the step are $i$ and $j$. We write $i, j$ instead than $\{i, j\}$, for simplicity.

*Example 1.* We now show the semantic of the example presented in the Introduction. Figure ?(a) shows the semantic of $S\{^a/_{sec}\}$, for simplicity we do not write neither the restriction on channels $c$ and $out$ neither the termination symbol $0$ of each component, furthermore, we use $-$ to denote a component that is stuck. Similarly the semantic of $S\{^a/_{sec}\}$ is shown in Figure ?(b).

The set of enable transitions also become clear in the figures, we have, for instance, $Enabled(S\{^b/_{sec}\}) = \{(1,2),(1,3),(1,4)\}$ and $Enabled(\ 0 \parallel \overline{out}\langle a\rangle \parallel\ -\ \parallel -\ ) = \{2\}$. Finally, the scheduler $\zeta$ defined as

$$\zeta(\sigma) \stackrel{\text{def}}{=} \begin{cases} (1,4) & if\ \sigma = S\{^a/_{sec}\}, \\ 2 & if\ \sigma = S\{^a/_{sec}\} \xrightarrow{1,2:\tau} (\ 0 \parallel \overline{out}\langle a\rangle \parallel\ -\ \parallel -\ ), \\ 3 & if\ \sigma = S\{^a/_{sec}\} \xrightarrow{1,3:\tau} (\ 0 \parallel\ -\ \parallel \overline{out}\langle b\rangle \parallel -\ ), \\ 4 & if\ \sigma = S\{^a/_{sec}\} \xrightarrow{1,4:\tau} (\ 0 \parallel\ -\ \parallel\ -\ \parallel \overline{out}\langle a\rangle\ ), \\ \bot & otherwiase, \end{cases}$$

is a global scheduler for $S\{^a/_{sec}\}$.




1(a) $S\{^a/_{sec}\}$ 1(b) $S\{^b/_{sec}\}$

**Mig:** I am having some issues with the numbering of this figure, try to solve it, otherwise insert the figure number explicitly in the text

## 4 Safe equivalences

In this section we revise some of the main equivalence notions used in literature so to guarantee their safe use in security.

In the following we assume that any probabilistic automaton is unfolded into a tree (this is a standard construction, see for instance [**?**]). This way, every finite path $\sigma$ is determined by its last state, $last(\sigma)$. Consequently, we can view a scheduler as a function on states, and we write $\zeta(q)$ instead than $\zeta(\sigma)$, where $q = last(\sigma)$.

### 4.1 Safe Traces

We define here a safe version of complete-trace semantics. The idea is that we want to compare two processes on the basis not only of their traces, but also of the choices that the global scheduler makes at every step. One way of doing this is by recording explicitly the tags in the traces.

**Definition 3.**

– *Given a TAP $M = (Q, L, \Sigma, \hat{q}, \alpha)$ the (complete) safe traces of $M$, denoted here by $Traces_s(M)$, are defined as the sequence of tags and actions in all possible complete executions, i.e.*

$$\begin{array}{ll} Traces_s(M) = & \\ \quad \{\ell_1 : a_1 \cdot \ell_2 : a_2 \cdot \ldots \cdot \ell_n : a_n \mid \hat{q} \xrightarrow{\ell_1:a_1} q_1 \xrightarrow{\ell_2:a_2} \ldots \xrightarrow{\ell_n:a_n} q_n \not\rightarrow\} & \textit{finite} \\ \quad \cup & \\ \quad \{\ell_1 : a_1 \cdot \ell_2 : a_2 \cdot \ldots \cdot \ell_n : a_n \cdot \ldots \mid \hat{q} \xrightarrow{\ell_1:a_1} q_1 \xrightarrow{\ell_2:a_2} \ldots \xrightarrow{\ell_n:a_n} q_n \ldots\} & \textit{infinite} \end{array}$$

– *Given a system $q$, we will denote by $Traces_s(q)$ the safe traces of the automaton associated to $q$.*
– *Two systems $q_1$ and $q_2$ are safe-trace equivalent, denoted by $q_1 \simeq_s q_2$, if and only if $Traces_s(q_1) = Traces_s(q_2)$.*

It is clear that safe-trace equivalence is at least as discriminating as the standard (complete-) trace equivalence, denoted here by $\simeq$, which compares only the sequences of actions. In fact, the latter is obtained from the former by abstracting from the tags. The following example points out the converse does not hold.

*Example 2.* Consider the system $S$ given in the introduction. There we have

$$Traces(S\,\{^a/_{sec}\}) = \{a, b\} = Traces(S\,\{^b/_{sec}\}).$$

On the other hand we have

$$Traces_s(S\,\{^a/_{sec}\}) = \{1, 2 : \tau \cdot 2 : a, 1, 3 : \tau \cdot 3 : b, 1, 4 : \tau \cdot 4 : a\}$$

$$\neq$$

$$Traces_s(S\,\{^b/_{sec}\}) = \{1, 2 : \tau \cdot 2 : a, 1, 3 : \tau \cdot 3 : b, 1, 4 : \tau \cdot 4 : b\}.$$

### 4.2 Safe Bisimilarity

In this section we propose a security-safe version of bisimulation, that we call *safe bisimulation*. This is an equivalence relation stricter than safe-trace equivalence, with the advantage of being a congruence.

We start with some notation. Given a TPA $M = (Q, L, \Sigma, \hat{q}, \alpha)$, and a global scheduler $\zeta$, we denote by $\alpha_\zeta$ the restriction of $\alpha$ to $\zeta$, i.e. for every $q \in Q$,

$$\alpha_\zeta(q) = \{(a, \mu) \mid \text{there exists } \ell \in L \text{ such that } (\ell, a, \mu) \in \alpha(q) \text{ and } \zeta(q) = \ell\}$$

We will also write $q \xrightarrow{a}_\zeta \mu$ for $(a, \mu) \in \alpha_\zeta(q)$, and $M_\zeta$ for the automaton obtained by pruning $M$ from all the choices not compatible with $\zeta$, i.e. $M_\zeta = (Q, L, \Sigma, \hat{q}, \alpha_\zeta)$. Note that $M_\zeta$ still contains nondeterminism, since there may be $\mu_1, \mu_2$, with $\mu_1 \neq \mu_2$, such that $(a_1, \mu_1), (a_2, \mu_2) \in \alpha_\zeta$ (with either $a_1 = a_2$ or $a_1 \neq a_2$).

We now define the notion of safe bisimulation. The idea is that, if $q$ and $q'$ are bisimilar states, then every move from $q$ should be mimicked by a move from $q'$ *using the same scheduler*.

**Definition 4.** *Given a TPA $M = (Q, L, \Sigma, \hat{q}, \alpha)$, we say that a relation $\mathcal{R} \subseteq Q \times Q$ is a safe bisimulation if, whenever $q_1 \mathcal{R} q_2$, then, for all global schedulers $\zeta$ for $M$:*

- *if $q_1 \xrightarrow{a}_\zeta \mu_1$, then there exists $\mu_2$ such that $q_2 \xrightarrow{a}_\zeta \mu_2$, $\zeta(q_1) = \zeta(q_2)$, $\mu_1 \mathcal{R} \mu_2$, and*
- *if $q_2 \xrightarrow{a}_\zeta \mu_2$, then there exists $\mu_1$ such that $q_1 \xrightarrow{a}_\zeta \mu_1$, $\zeta(q_1) = \zeta(q_2)$, $\mu_1 \mathcal{R} \mu_2$.*

*where $\mu_1 \mathcal{R} \mu_2$ means that for all equivalence classes $X \in Q/\mathcal{R}$, we have $\mu_1(X) = \mu_2(X)$.*

The following result is immediate:

**Proposition 1.** *The union of all the safe bisimulations for $M$ is still a safe bisimulation for $M$.*

Therefore the largest safe bisimulation exists, and coincides with the union of all safe bisimulations. We call it *safe bisimilarity*, and we denote it by $\sim_s$.

Given two TPA's on the same $L$ and $\Sigma$, $M_1 = (Q_1, L, \Sigma, \hat{q}_1, \alpha_1)$ and $M_2 = (Q_2, L, \Sigma, \hat{q}_2, \alpha_2)$, we can define bisimulation and bisimilarity across their states, i.e. as relations on $(Q_1 \cup Q_2)$, in the obvious way, by constructing the TPA $M$ with a new initial state $\hat{q}$ and two transitions to $\delta_{\hat{q}_1}$ and to $\delta_{\hat{q}_2}$, respectively.

Given two components or systems, $q_1$ and $q_2$, we will say that $q_1$ and $q_2$ are safely bisimilar, denoted by $q_1 \sim_s q_2$, if the initial states of the corresponding TPA's are safely bisimilar. Note that $q_1 \sim_s q_2$ is possible only if $q_1$ and $q_2$ have the same number of active components, where "active", for a component, means that during the execution of the system it will make at least one step.

Note that in the case of components, or of systems constituted by one component only, safe bisimulation and safe bisimilarity coincide with standard bisimulation and bisimilarity, respectively. For systems, safe bisimulation is at least as strong as standard bisimulation (denoted by $\sim$):

*Remark 1.* Given two systems $q_1$ and $q_2$, if $q_1 \sim_s q_2$ then $q_1 \sim q_2$.

The converse does not hold in general, as shown by the following example.

*Example 3.* We consider again the system $S$ presented in the Introduction. It is easy to see that $S\{^a/_{sec}\} \sim S\{^b/_{sec}\}$. In order to show that $S\{^a/_{sec}\} \not\sim_s S\{^b/_{sec}\}$ we take the two automata of $S\{^a/_{sec}\}$ and $S\{^b/_{sec}\}$ (see Figure ?), and construct a new automaton (as described above) with initial state $\hat{q}$ such that $\hat{q} \xrightarrow{\ell:\tau} S\{^a/_{sec}\}$ and

$\hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{b}/_{sec}\right\}$. Now consider the scheduler $\zeta$ such that

$$\zeta(\sigma) \stackrel{\text{def}}{=} \begin{cases} \ell & if\ \sigma = \hat{q}, \\ (1,4) & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{a}/_{sec}\right\}, \\ 2 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{a}/_{sec}\right\} \xrightarrow{1,2:\tau} (\ 0 \mid\mid \overline{out}\langle a\rangle \mid\mid\ - \mid\mid -\ ), \\ 3 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{a}/_{sec}\right\} \xrightarrow{1,3:\tau} (\ 0 \mid\mid\ - \mid\mid \overline{out}\langle b\rangle \mid\mid -\ ), \\ 4 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{a}/_{sec}\right\} \xrightarrow{1,4:\tau} (\ 0 \mid\mid\ - \mid\mid\ - \mid\mid \overline{out}\langle a\rangle\ ), \\ (1,4) & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{b}/_{sec}\right\}, \\ 2 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{b}/_{sec}\right\} \xrightarrow{1,2:\tau} (\ 0 \mid\mid \overline{out}\langle a\rangle \mid\mid\ - \mid\mid -\ ), \\ 3 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{b}/_{sec}\right\} \xrightarrow{1,3:\tau} (\ 0 \mid\mid\ - \mid\mid \overline{out}\langle b\rangle \mid\mid -\ ), \\ 4 & if\ \sigma = \hat{q} \xrightarrow{\ell:\tau} S\left\{{}^{b}/_{sec}\right\} \xrightarrow{1,4:\tau} (\ 0 \mid\mid\ - \mid\mid\ - \mid\mid \overline{out}\langle b\rangle\ ), \\ \bot & otherwiase. \end{cases}$$

It is easy to see that, under $\zeta$, $S\left\{{}^{b}/_{sec}\right\}$ cannot simulate the transition $4:a$ produced by $S\left\{{}^{a}/_{sec}\right\}$.

It turns out that safe bisimulation is a congruence with respect to all the operators of our language, as expressed by the following theorem. Note that the first two items are just the standard compositionality result for probabilistic bisimulation.

**Theorem 1.** *Let $a \in \Sigma$ and $A, B, B' \subseteq \Sigma$. Let $p_1, \ldots, p_n$ be probability values, and $q, q_1, q_2, \ldots, q_n, q'_1, q'_2, \ldots, q'_n$ be components.*

- *If $q_1 \sim_s q_2$, then $a.q_1 \sim_s a.q_2$, $q_1 + q \sim_s q_2 + q$, and $(a)q_1 \sim_s (a)q_2$.*
- *If $q_1 \sim_s q'_1, \ldots, q_n \sim_s q'_n$, then $\sum_i p_i : q_i \sim_s \sum_i p_i : q'_i$.*
- *If $(B)\ q_1 \parallel \ldots \parallel q_n\ \sim_s\ (B')\ q'_1 \parallel \ldots \parallel q'_n$, then*

$$(A \cup B)\ q_1 \parallel \ldots \parallel q \parallel \ldots \parallel q_n\ \sim_s\ (A \cup B')\ q'_1 \parallel \ldots \parallel q \parallel \ldots \parallel q'_n.$$

The following property shows that bisimulation is stronger than safe-trace equivalence, like in the standard case.

**Proposition 2.** *For every pair of components or systems, $q_1$ and $q_2$, if $q_1 \sim_s q_2$ then $q_1 \simeq_s q_2$.*

Like in the standard case, the vice-versa does not hold, and safe-trace equivalence is not a congruence.

## 5 Admissible schedulers

In this section we restrict the discerning power of the global and local schedulers in order to avoid the problem of the information leakage induced in security by clairvoyant schedulers. We impose two kinds of restrictions: For the global scheduler, following

the framework proposed in [**?**], we assume that it can only see, and keep memory of, the observable actions and the components that are enabled. It cannot see the secret actions and the internal choices of the various components. As for the local scheduler, we assume that the local nondeterminism of each component is solved on the basis of the local view of the history (local to that component), i.e. the projection of the history of the system on that component. In other words, each component has to make decisions based only on the history of its own execution; it cannot see anything of the other components.

### 5.1 Restricting Global Schedulers

We assume that the set of actions $\sigma$ is divided in two parts, the *secret actions* $\mathcal{S}$ and the *observable actions* $\mathcal{O}$. The secret actions are supposed to be invisible to the global scheduler. Formally, this can be achieved using a function $sift$ defined as:

$$sift(a) = \begin{cases} \tau \text{ if } a \in \mathcal{S} \\ a \text{ otherwise} \end{cases}$$

Then, we restrict the power of the global scheduler by forcing it to make the same decisions on paths he cannot tell apart, as formalized in the next definition.

**Mig:** Try to link to dolev-yao

**Definition 5.** *Given a TPA $M$, a global scheduler $\zeta$ for $M$ is admissible if for all paths $\sigma_1$ and $\sigma_2$ we have*

$$t(\sigma_1) = t(\sigma_2) \quad \textit{implies} \quad \zeta(\sigma_1) = \zeta(\sigma_2)$$

*where*

$$t\left(\hat{q} \xrightarrow{l_1:a_1} q_1 \xrightarrow{l_2:a_2} \cdots \xrightarrow{l_n:a_n} q_{n+1}\right) \stackrel{\text{def}}{=} \begin{array}{r} (Enabled(q_0), sift(a_1), l_1) \\ (Enabled(q_1), sift(a_2), l_2) \\ \vdots \\ (Enabled(q_n), sift(a_n), l_n) \end{array}$$

The idea is that $t$ sifts the information of the path that the scheduler can see. This way, since $sift$ "hides" the secrets to the scheduler, the scheduler cannot take different decisions based on secret information.

### 5.2 Restricting Local Schedulers

The restriction on the local scheduler is based on the idea that a step of the component $i$ of a system can only be based on the view that $i$ has of the history, i.e. its own history. In order to formalize this restriction, it is convenient to introduce the concept of $i$-view of a path $\sigma$, or *projection* of $\sigma$ on $i$, which we will denote by $\sigma_{\restriction i}$. We define it inductively:

$$(\sigma \xrightarrow{\ell:a} \mu)_{\restriction i} = \begin{cases} \sigma_{\restriction i} \xrightarrow{i:b} \delta_{q_i} & \text{if } \ell = \{i, j\} \text{ and } \mu = \delta_{(A)\ q_1 \| \ldots \| q_i \| \ldots \| q_j \| \ldots \| q_n} \\ \sigma_{\restriction i} \xrightarrow{i:a} \mu & \text{if } \ell = i \\ \sigma_{\restriction i} & \text{otherwise} \end{cases}$$

In the above definition, the first line represents the case of a synchronization step involving the component $i$, where we assume that the premise for $i$ is of the form $q_i' \xrightarrow{b} \delta_{q_i}$. The second line represents an interleaving step in which $i$ is the active component. The third line represents step in which the component $i$ is idle.

**Cat:** This is a bit of cheating. In the journal version we need to change the operational semantics so that we can retrieve the form of the premise from the conclusion

The restriction to the local scheduler can now be expressed as follows:

**Definition 6.** *Given a TPA $M$ and a local scheduler $\xi$ for $M$, we say that $\xi$ is admissible if for all paths $\sigma$ and $\sigma'$, if $\xi(\sigma) = (\ell, a, \mu)$, and $\xi(\sigma') = (\ell', a', \mu')$:*

- *if $\ell = \ell' = i$ and $\sigma_{\uparrow i} = \sigma'_{\uparrow i}$, then $\xi(\sigma) = \xi(\sigma')$,*
- *if $\ell = \ell' = \{i, j\}$, $\sigma_{\uparrow i} = \sigma'_{\uparrow i}$, and $\sigma_{\uparrow j} = \sigma'_{\uparrow j}$ then $\xi(\sigma) = \xi(\sigma')$.*

A pair of compatible schedulers $(\zeta, \xi)$ for $M$ is called *admissible* if both $\zeta$ and $\imath$ are admissible.

## 6 Safe Nondeterministic Information Hiding

In this section we define the notion of information hiding under the most general hypothesis that the nondeterminism is handled partly in a demonic way and partly in an angelic way. We assume that the demonic part is in the realm of the global scheduler, while the angelic part is controlled by the local scheduler. The motivation is that in a protocol the local components can be thought of as programs running locally in a single machine, and locally predictable and controllable, while the network can be subject to attacks that make the interactions unpredictable.

We recall that, in a purely probabilistic setting, the absence of leakage is expressed as follows (see for instance [**?**]). Given a purely probabilistic automaton $M$, and a sequence $\tilde{a} = a_1 a_2 \ldots a_n$, let $\mathbf{P}_M([\tilde{a}])$ represent the probability measure of all complete paths with trace $\tilde{a}$ in $M$. Let $P$ be a protocol containing a variable action $secr$, and let $s$ be secret actions. Let $M(s)$ be the automaton corresponding to $P[s/secr]$. Define $Pr(\tilde{a} \mid s)$ as $\mathbf{P}_{M(s)}([\tilde{a}])$. Then $P$ is leakage-free if for every observable trace $\tilde{a}$ , and for every secret $s_1$ and $s_2$, we have:

$$Pr(\tilde{a} \mid s_1) = Pr(\tilde{a} \mid s_2).$$

In a purely nondeterministic setting, on the other hand, the absence of leakage has been characterized in the literature by the following property:

$$P[s_1/secr] \cong P[s_2/secr]$$

where $\cong$ is an equivalence relation like trace equivalence, or bisimulation. As we have argued in the introduction, this definition assumes an angelic interpretation of nondeterminism.

We want to combine the above notions so to come with the case in which we have both probability and nondeterminism. Furthermore, we want to extend it to the case in which part of the nondeterminism is interpreted demonically. Let us first introduce some notation.

Let $S$ be a system containing a variable action $secr$. Let $s$ be a secret action. Let $M(s)$ be the TPA associated to $S[s/secr]$ and let $(\zeta, \xi)$ be a compatible pair of global and local schedulers for $M(s)$. The probability of an observable trace $\tilde{a}$, given $s$, is defined as

$$Pr_{\zeta,\xi}(\tilde{a} \mid s) = \mathbf{P}_{M(s),\zeta,\xi}([\tilde{a}]).$$

The global nondeterminism is interpret demonically, and therefore we need to ensure that the conditional of an observable, given the two secrets, are calculated with respect to the same global scheduler. However, the scheduler should not be too powerful, i.e. we want to rule out the possibility that the scheduler use the secret information (i.e. be clairvoyant ) to accomplish its demonic goals.

**Definition 7.** *A system $S$ is leakage-free if, for every pair of secrets $s_1$ and $s_2$, and every admissible scheduler $\zeta$,*

$$\{Pr_{\zeta,\xi}(\tilde{a} \mid s_1) \mid \xi \textit{ compatible with } \zeta\} = \{Pr_{\zeta,\xi}(\tilde{a} \mid s_2) \mid \xi \textit{ compatible with } \zeta\}.$$

It turns out that the safe equivalences defined in Section **??** imply the absence of leakage:

**Theorem 2.** *Let $S$ be a system with a variable action $secr$ and assume that $S[s_1/secr] \simeq S[s_2/secr]$ for every pair of secrets $s_1$ and $s_2$. Then $S$ is leakage-free.*

From the above theorem and from Propositiion **??**, we also have the following corollary (with the same premises as the previous theorem):

**Corollary 1.** *If $S[s_1/secr] \simeq S[s_2/secr]$ for every pair of secrets $s_1$ and $s_2$, then $S$ is leakage-free.*

## 7 Conclusion