Formalizing Probabilistic Noninterference

Andrei Popescu, Johannes Hölzl, and Tobias Nipkow

Technische Universität München

Abstract. We present an Isabelle formalization of probabilistic noninterference for a multi-threaded language with uniform scheduling. Unlike in previous settings from the literature, here probabilistic behavior comes from both the scheduler and the individual threads, making the language more realistic and the mathematics more challenging. We study resumption-based and trace-based notions of probabilistic noninterference and their relationship, and also discuss compositionality w.r.t. the language constructs and type-system-like syntactic criteria. The formalization uses recent development in the Isabelle probability theory library.

1 Introduction

Language-based noninterference [25] is a major topic in computer security. To state noninterference, one typically assumes the program memory is separated into a *low*, or public, part, which an attacker is able to observe, and a *high*, or private, part, hidden to the attacker. A program satisfies noninterference if, upon running it, the high part of the initial memory does not affect the low part of the resulting memory. In other words, the program has *no information leaks* from the private part of the memory into the public one, so that a potential attacker should not be able to obtain information about private data by inspecting public data.

While research on language-based noninterference has been thriving in recent years, only little effort has been put in the mechanical verification of results in this area. In a previous paper [23], we presented a formalization of possibilistic noninterference, with a focus on compositionality and type-system-like syntactic criteria. Here, we continue this research agenda with noninterference for a probabilistic language. The general motivation for our formalization efforts is the belief, shared by more and more researchers lately, that the development of programming language metatheory should be pursued with the help (and confidence) offered by a proof assistant [1]. But there is also a more specific motivation. Previous work on probabilistic language-based noninterference is presented in a very informal fashion, even by the standards of a "pen-and-paper" mathematician. While justified by the complexity of the involved concepts, this situation is certainly not satisfactory. The work reported here tries to alleviate this problem, taking advantage of the recent development of a rich Isabelle/HOL library for probability theory [9, 10].

We start by formalizing a probabilistic multi-threaded language and its small-step operational semantics under a uniform scheduler (§2). Then we proceed with the formalization of noninterference properties (§3). At the heart of the formalization is an abstract equivalence $\sim$ on the memory states, called *indistinguishability*, where $s \sim t$ intuitively means that an attacker cannot distinguish between s and t. (For a concrete notion of state that assigns values to variables and a classification of variables as high or low, $\sim$ becomes the standard low equality, i.e., identity on the low variables.) In

this context, noninterference of a program roughly means that selected parts of its execution are compatible with $\sim$, in that if starting in indistinguishable states they yield indistinguishable results. We consider two flavors of noninterference.

Resumption-based (or *bisimulation-based*) noninterference (§3.1), amply represented in the literature [3–6, 26–31] requires that each execution chunk (where the chunk may be one single step or several steps, depending on the specific notion) is compatible with $\sim$ on states, *and that this property is also resumed in the matching continuations.* For a probabilistic language, it does not suffice to speak of solitary matching continuations; instead, one partitions the continuations and matches the sets of the partition in a probability-preserving way. A main advantage of resumption-based notions is usually compositionality w.r.t. the language constructs; as we argue in [23] for possibilistic noninterference, this can form the basis of the *automatic* inference of type-system-like criteria—and indeed, this also applies here (§3.3). Therefore, we take compositionality as a major test for a newly introduced resumption-based notion (§3.2). A first notion we consider is a variation of standard probabilistic bisimilarity, which is mostly compositional but does not interact well with thread-termination sensitive parallel composition. To cope with this problem, we define a weaker notion, 01-bisimilarity, relaxing the requirement to match continuation steps by allowing stutter moves.

Trace-based noninterference (§3.4), rather scarce in language-based settings [17,33] but pervasive in system-based settings (overviewed in [16]), requires that the whole set of execution traces is compatible with $\sim$. In a possibilistic framework, this would mean that, given two indistinguishable states $s \sim t$, for any execution starting in s and ending in s' there exists an execution starting in t and ending in some t' such that $s' \sim t'$. In a probabilistic framework however, one needs to take a global view and equate, for each possible indistinguishability class S of the result, the (cumulated) measure of all traces starting in s and ending in S with the measure of all traces starting in t and ending in S. We formalize two natural trace-based notions representing end-to-end security guarantees of our two main resumption-based notions.

The results of our formal development [21] can be summarized as follows:

$$\text{syntactic criteria} \overset{\text{compositionality}}{\Longrightarrow} \text{resumption noninterference} \Longrightarrow \text{trace noninterference}$$

Besides the certification aspect, our formalization makes new contributions to the state of the art in language-based noninterference:
– it considers for the first time a fully probabilistic language, where probabilistic behavior comes not only from the scheduler, but also from the individual threads (through probabilistic choice);
– it performs a comprehensive study, including both trace-based and resumption-based noninterference and their comparison.

On the other hand, the formalized language has several limitations:
– it restricts thread communication to shared-state communication;
– it does not cover dynamic thread creation;
– it is confined to a uniform scheduler, assigning equal probabilities to each thread.

Throughout the paper, we employ notations close to the formalization, but we occasionally take some liberties with the Isabelle notation in order to ease the presentation.

2 The programming language

We formalize a programming language featuring the usual sequential commands, extended with probabilistic choice and parallel composition under a uniform scheduler.

2.1 Syntax

The language is parameterized by the following types:
– **atom**, of atoms, ranged over by *atm*;
– **test**, of tests, ranged over by *tst*;
– **choice**, of (probabilistic) choices, ranged over by *ch*.

Standard examples of atoms and tests are assignments such as $x := x+y$ and Boolean expressions such as $x < y+x$. Moreover, as we discuss in §2.2, choices are flexible enough to cover the standard "if" conditions, as well as stateless probabilistic choice.

The type **com**, of commands, ranged over by c, d, is defined as follows:

$$\begin{array}{l}\texttt{datatype}\ \textbf{com} = \mathsf{Atm}\ \textbf{atom} \mid \mathsf{Done} \mid \mathsf{Seq}\ \textbf{com}\ \textbf{com} \mid \mathsf{While}\ \textbf{test}\ \textbf{com} \mid \\ \qquad\qquad \mathsf{Ch}\ \textbf{choice}\ \textbf{com}\ \textbf{com} \mid \mathsf{Par}\ (\textbf{com list}) \mid \mathsf{ParT}\ (\textbf{com list})\end{array}$$

For atomic commands $\mathsf{Atm}\ atm$ we usually omit the constructor Atm. The lists of commands passed as arguments to Par and ParT will be indicated using explicit index notation, e.g., $[c_0, \ldots, c_{n-1}]$ is a list of length n—this is a detour from the Isabelle syntax aimed at making the presentation clearer.

A command is called *finished* if it is either Done or, inductively, a Par- or ParT-composition of finished commands: $\mathsf{finished}\ \mathsf{Done}$;
$(\bigwedge_{i=0}^{n-1} \mathsf{finished}\ c_i) \Longrightarrow \mathsf{finished}\ (\mathsf{Par}\ [c_0, \ldots, c_{n-1}])$;
$(\bigwedge_{i=0}^{n-1} \mathsf{finished}\ c_i) \Longrightarrow \mathsf{finished}\ (\mathsf{ParT}\ [c_0, \ldots, c_{n-1}])$.

$\mathsf{Seq}\ c_1\ c_2$ is the sequential composition of c_1 and c_2, written in concrete syntax[1] as $c_1\ ;\ c_2$. $\mathsf{While}\ tst\ c$ is the usual while loop, in concrete syntax, while *tst* do *c*. $\mathsf{Ch}\ ch\ c_1\ c_2$ is a choice command. $\mathsf{Par}\ [c_0, \ldots, c_{n-1}]$ and $\mathsf{ParT}\ [c_0, \ldots, c_{n-1}]$ are two variants of parallel composition of the thread pool $[c_0, \ldots, c_{n-1}]$, written in concrete syntax as $c_1 \parallel \ldots \parallel c_{n-1}$ and $c_1 \parallel_T \ldots \parallel_T c_{n-1}$, respectively. They differ in that the latter is termination-sensitive, removing finished threads from the thread pool.

2.2 Semantics

The semantics of the language indicates the immediate steps available to a command in a given state, where the steps are assigned weights that sum up to 1. It is parameterized by the following data:
– a type of (memory) states, **state**, ranged over by s, t;
– an execution function for the atoms, $\mathsf{aexec} : \textbf{atom} \to \textbf{state} \to \textbf{state}$;
– an evaluation function for the tests, $\mathsf{tval} : \textbf{test} \to \textbf{state} \to \textbf{bool}$;
– an evaluation function for the choices, $\mathsf{cval} : \textbf{choice} \to \textbf{state} \to [0,1]$, where $[0,1]$ is the real unit interval.

$\mathsf{cval}\ ch\ s$ expresses the probability with which the left branch, c_1, will be picked when executing the command $\mathsf{Ch}\ ch\ c_1\ c_2$ (while the right branch, c_2, will be picked with probability $1 - \mathsf{cval}\ ch\ s$).

[1] We use abstract syntax in theoretical results and concrete syntax in examples.

c	$\mathsf{wt}\ c\ s\ i$	$\mathsf{cont}\ c\ s\ i$	$\mathsf{eff}\ c\ s\ i$
atm	1	Done	$\mathsf{aexec}\ c\ s$
Done	1	Done	s
$\mathsf{Seq}\ c_1\ c_2$	$\mathsf{wt}\ c_1\ s\ i$	c_2, if finished c_1 $\mathsf{Seq}\ (\mathsf{cont}\ c_1\ i)\ c_2$, otherwise	$\mathsf{eff}\ c_1\ s\ i$
$\mathsf{Ch}\ ch\ c_1\ c_2$	$\mathsf{cval}\ ch\ s$, if $i=0$ $1-\mathsf{cval}\ ch\ s$, if $i=1$	c_1, if $i=0$ c_2, if $i=1$	s
$\mathsf{While}\ tst\ d$	1	$\mathsf{Seq}\ d\ (\mathsf{While}\ tst\ d)$, if $\mathsf{tval}\ tst\ s$ Done, otherwise	s
$\mathsf{Par}\ [c_0,\ldots,c_{n-1}]$	$\frac{1}{n} * \mathsf{wt}\ c_k\ s\ j$	$\mathsf{Par}\ [c_0,\ldots,\mathsf{cont}\ c_k\ s\ j,\ldots,c_{n-1}]$	$\mathsf{eff}\ c_k\ s\ j$
$\mathsf{ParT}\ [c_0,\ldots,c_{n-1}]$	$\frac{1}{m} * \mathsf{wt}\ c_k\ s\ j$, if $\neg\ \mathsf{finished}\ c_k$ 0, otherwise	$\mathsf{ParT}\ [c_0,\ldots,\mathsf{cont}\ c_k\ s\ j,\ldots,c_{n-1}]$	$\mathsf{eff}\ c_k\ s\ j$

$(k,j) \equiv$ the unique pair in $\mathbf{nat}\times\mathbf{nat}$ such that $0\le k<n \wedge 0\le j<\mathsf{brn}\ c_k \wedge i=(\sum_{l=0}^{k-1}\mathsf{brn}\ c_l)+j$

$m \equiv$ the number of indexes $l\in\{0,\ldots,n-1\}$ such that $\neg\ \mathsf{finished}\ c_l$

Fig. 1: Probabilistic Small-Step Semantics

For every command c, we define its *branching number* (*branching* for short), $\mathsf{brn}\ c$: $\mathsf{brn}\ atm = \mathsf{brn}\ \mathsf{Done} = \mathsf{brn}\ (\mathsf{While}\ tst\ d) = 1$; $\mathsf{brn}\ (\mathsf{Seq}\ c_1\ c_2) = \mathsf{brn}\ c_1$; $\mathsf{brn}\ (\mathsf{Ch}\ ch\ c_1\ c_2) = 2$; $\mathsf{brn}\ (\mathsf{Par}\ [c_0,\ldots,c_{n-1}]) = \mathsf{brn}\ (\mathsf{ParT}\ [c_0,\ldots,c_{n-1}]) = \sum_{l=0}^{n-1}\mathsf{brn}\ c_l$.

Indexes ranging from 0 to $\mathsf{brn}\ c-1$ are used to label the single-step transitions available from a command c. Then, given current states s, these transitions are assigned *weights*, *continuation commands* (*continuations* for short) and *state effects* (*effects* for short) by the functions $\mathsf{wt}\ c\ s : \{0,\ldots,\mathsf{brn}\ c-1\}\to[0,1]$, $\mathsf{cont}\ c\ s : \{0,\ldots,\mathsf{brn}\ c-1\}\to$ **com**, and $\mathsf{eff}\ c\ s : \{0,\ldots,\mathsf{brn}\ c-1\}\to$ **state**, respectively. All these are defined in Fig. 1's table. The first column lists all possible forms of c, and the other columns show, for the three operators, the defining recursive clauses for each form (with $i\in\{0,\ldots,\mathsf{brn}\ c-1\}$).

The semantics of *atm* and Done are straightforward: there is one single available step (hence brn is 1) with weight 1, transiting to the terminating continuation Done. For Done, there is no effect on the state (i.e., the state s remains unchanged), and for *atm*, the effect is given by aexec. For technical reasons, Done has a dummy transition to itself.

A $\mathsf{Seq}\ c_1\ c_2$ command obtains its branching, weight and effect from c_1, and its continuation is the continuation of c_1 if unfinished and is c_2 otherwise. $\mathsf{While}\ tst\ d$ performs an unfolding step to the continuation $\mathsf{Seq}\ d\ (\mathsf{While}\ tst\ d)$ if the test is True and to Done otherwise, in both cases with weight 1 and no effect.

A choice command $\mathsf{Ch}\ ch\ c_1\ c_2$ is assumed to perform an effectless branching according to ch. It has 2 branches, labeled 0 and 1: the left one with weight $\mathsf{cval}\ ch\ s$ and continuation c_1, and the right one with complementary weight $1-\mathsf{cval}\ ch\ s$ and continuation c_2.

If c has the form $\mathsf{Par}\ [c_0,\ldots,c_{n-1}]$, then it consists of n threads, $c_0,\ldots,c_{n-1}$, running in parallel under a uniform scheduler assigning them equal probabilities. The branching of c is thus the sum the branchings of c_l for $l\in\{0,\ldots,n-1\}$. A branch label i of c determines uniquely the numbers k and j so that i corresponds to the j's branch of c_k,

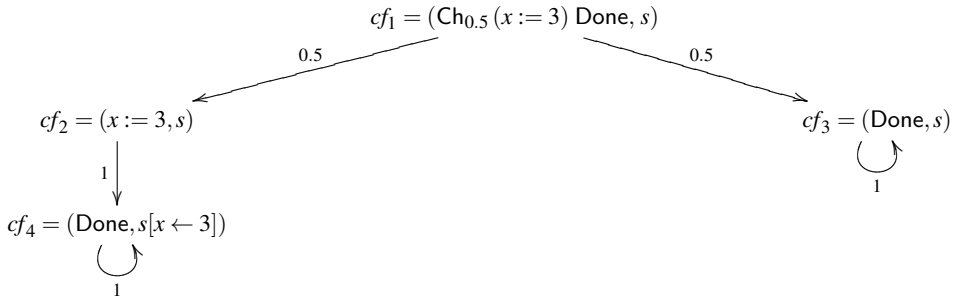

Fig. 2: Markov Chain

via the equation $i = (\sum_{l=0}^{k-1} \mathsf{brn}\ c_l) + j$. The weight of i in c is [the probability of picking thread c_k (out of n possibilities)] times [the weight of j in c_k], i.e., $(1/n) * \mathsf{wt}\ c_k\ s\ j$. The i-continuation from c is obtained by replacing, in the thread pool, c_k with its j-continuation $\mathsf{cont}\ c_k\ s\ j$. The i-effect of c is the j-effect of c_k, $\mathsf{eff}\ c_k\ s\ j$.

ParT behaves like Par, except that it is termination-sensitive, in that finished threads are not taken into consideration (being assigned weight 0), and the choice is made among the m unfinished threads—consequently, the weight of an unfinished thread is $1/m$. In case all threads are finished, we simply add a single idle transition, with probability 1—this trivial case is not shown in the figure.

Example 1 Here is a simple standard instantiation of the generic notions of state, atomic statement and test. **state** consists of assignments of values to variables, **var** $\to$ **val**, where **val** is a type of values (e.g., integers) and **var** a countable type of variables. The atomic statements and the tests are built by means of arithmetic and boolean expressions applied to variables. The atom and test valuation functions are as expected.

Since the choice is allowed to depend on the state, it can capture not only standard probabilistic choice, but also the "if" statement [11]. We define **choice** to be $[0,1] +$ **test**, i.e., a choice is either $\mathsf{Inl}\ x$, the embedding of a unital real number, or $\mathsf{Inr}\ tst$, the embedding of a test tst. Then $\mathsf{cval}\ (\mathsf{Inl}\ x)\ s = x$ for all s, making $\mathsf{Ch}\ (\mathsf{Inl}\ x)\ c_1\ c_2$, simply written $\mathsf{Ch}_x\ c_1\ c_2$, the stateless probabilistic choice. Moreover, $\mathsf{cval}\ (\mathsf{Inr}\ tst)\ s = 1$ if $\mathsf{tval}\ tst\ s = \mathsf{True}$ and $= 0$ otherwise, making $\mathsf{Ch}\ (\mathsf{Inr}\ tst)\ c_1\ c_2$, simply written $\mathsf{if}\ tst\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$, the standard conditional statement.

For any command, the sum of the weights of its branches is 1, meaning that the small-step semantics yields the transition matrix M of a Markov chain on the type of configurations, **config** = **com** $\times$ **state**:

$$\mathsf{M}\ (c,s)\ (c',s') = \sum \{\mathsf{wt}\ c\ s\ i \mid i \in \{0,\ldots,\mathsf{brn}\ c-1\} \wedge (c',s') = (\mathsf{cont}\ c\ s\ i, \mathsf{eff}\ c\ s\ i)\}$$

Fig. 2 shows the portion of the Markov chain reachable from $(\mathsf{Ch}_{0.5}\,(x := 3)\ \mathsf{Done}, s)$. Note that a node (c,s) may have fewer outer edges than $\mathsf{brn}\ c$, since some branches may lead to the same node, case in which they are merged into a single transition weighed with the sum of their weights. E.g., if $c = \mathsf{Ch}_{0.5}\,d\ d$, then (c,s) has a single Markov transition to (d,s) of weight $0.5 + 0.5 = 1$.

Let $\mathsf{Trace}_{(c,s)} = \{(c_i,s_i)_{i\in\mathbf{nat}} \mid (c_0,s_0) = (c,s) \wedge \forall i \in \mathbf{nat}.\, \mathsf{M}\,(c_i,s_i)\,(c_{i+1},s_{i+1}) > 0\}$, the set of *traces starting at* (c,s) (*(c,s)-traces* for short). A basic event for (c,s) is the set of all (c,s)-traces of a given finite prefix $(c_i,s_i)_{i=0}^{n}$; the measure of such a basic event is the product of transition weights $\prod_{i=0}^{n} \mathsf{M}\,(c_i,s_i)\,(c_{i+1},s_{i+1})$. Let $\mathsf{Alg}_{(c,s)}$ be the σ-algebra generated by the basic events, i.e., the smallest collection of subsets of $\mathsf{Trace}_{(c,s)}$ that is closed under countable union and complement and contains every basic event. By standard probability theory [13], there is a unique probability measure $\Pr_{(c,s)} : \mathsf{Alg}_{(c,s)} \to [0,1]$ extending the measure of basic events. ([9] describes in detail the formalization of the involved standard constructions.) For example, in Fig. 2's Markov chain, we have only two (c_0,s)-traces, $cf_1\,cf_2\,cf_4^{\omega}$ and $cf_1\,cf_3^{\omega}$. In general, the set of traces may be infinite, even uncountable. We have $\Pr_{(c_0,s)}\{cf_1\,cf_2\,cf_4^{\omega}\} = \Pr_{(c_0,s)}\{cf_1\,cf_3^{\omega}\} = 0.5$.

3 Noninterference

We fix a relation $\sim$ on states, called *indistinguishability*, where $s \sim t$ is meant to say "s and t are indistinguishable by the attacker."

Example 2 In the context of Example 1, $\sim$ is often defined as follows. Variables are classified as either low (lo) or high (hi) by a given security level function $\mathsf{sec} : \mathbf{var} \to \{\mathsf{lo},\mathsf{hi}\}$. Then $\sim$ is defined as coincidence on the low variables, with the intuition that the attacker can only observe these: $s \sim t \equiv \forall x \in \mathbf{var}.\, \mathsf{sec}\,x = \mathsf{lo} \Longrightarrow s\,x = t\,x$.

Noninterference of a program states that its execution is compatible with the indistinguishability relation: given two indistinguishable states s and t, (partially) executing the program once starting from s and once starting from t yield indistinguishable states.

There are two main types of formulation of noninterference: as an indefinite indistinguishability resumption property (bisimilulation) and as a property of alternative execution traces. The seminal paper [32] proposes an end-to-end noninterference property using big-step semantics, which can be seen as a property of traces. Much of subsequent work [3–6, 26–31] prefers small-step semantics and resumption-based notions, although trace-based notions are also considered [17, 33].

In the presence of concurrency, resumption-based notions have been shown to be more compositional (which was no surprise, since this phenomenon is known from process algebra). Sabelfeld and Sands [26] were the first to observe the tight connection between the compositionality of resumption-based notions and sufficient type-system criteria—in a previous paper [23], we used this idea to devise a uniform methodology for extracting syntactic criteria from compositionality.

On the other hand, trace-based notions are often more intuitive to grasp, as they do not involve the alternation complexity of bisimulations. Also, trace-based notions can benefit from other other kinds of static analyses, such as data-race analysis [33].

Next, we study and relate the two flavors of noninterference, including compositionality and syntactic criteria, for the introduced probabilistic language.

3.1 Resumption-Based Noninterference

We define the following notions of self isomorphism, siso, and discreetness, discr, *coinductively as greatest fixed points*, i.e., as the *weakest* predicates satisfying certain equations. (They are probabilistic counterparts of possibilistic notions introduced in [23].)

For siso, one requires that, if started in indistinguishable states, executions take the same branches *with the same probabilities*. For discr, one requires that, during the computation, the states stay indistinguishable from the initial state.

$\mathsf{siso}\ c \equiv (\forall s\ t\ i.\ s \sim t\ \wedge\ i < \mathsf{brn}\ c \Longrightarrow \mathsf{cont}\ c\ s\ i = \mathsf{cont}\ c\ t\ i\ \wedge\ \mathsf{eff}\ c\ s\ i \sim \mathsf{eff}\ c\ t\ i) \wedge$
$\qquad (\forall s\ i.\ i < \mathsf{brn}\ c \Longrightarrow \mathsf{siso}\ (\mathsf{cont}\ c\ s\ i))$

$\mathsf{discr}\ c \equiv \forall s\ i.\ i < \mathsf{brn}\ c \Longrightarrow s \sim \mathsf{eff}\ c\ s\ i\ \wedge\ \mathsf{discr}\ (\mathsf{cont}\ c\ s\ i)$

siso and cont are very demanding notions of security. To define weaker notions, we need to allow alternative executions to take different branches, while also allowing execution to change the indistinguishability class of the state. It is easy to notice that these two relaxations lead us to the consideration of (binary) relations rather than unary predicates. Indeed, if the command c branches according to a high test in two continuations d_1 and d_2, then the notion of security of c is conditioned by the notion of "equivalence" of d_1 and d_2. This equivalence will be defined as bisimilarity, while security of c will be defined as self bisimilarity (c bisimilar to itself).

To introduce probabilistic bisimulation, we need a few preparations. Given $I \subseteq \{0,\ldots,\mathsf{brn}\ c-1\}$, we write $\mathsf{Wt}\ c\ s\ I$ for the cumulated weights from (c,s) of the labels in I, namely, $\sum_{i\in I} \mathsf{wt}\ c\ s\ i$. Given sets A and P, we say P is a *partition* of A, written $\mathsf{part}\ A\ P$, if P consists of mutually disjoint sets whose union is A.

The following predicate $\mathsf{match}^{\mathsf{c}}_{\mathsf{c}}$ (read "match continuation against continuation") shows, for a relation on commands θ and two commands c and d, how the steps taken by c and d are matched unambiguously and weight-exhaustively, so that their effects are indistinguishable and their continuations are in θ:

$\mathsf{match}^{\mathsf{c}}_{\mathsf{c}}\ \theta\ c\ d \equiv$
$\forall s\ t.\ s \sim t \Longrightarrow \exists P\ Q\ F.$
$\mathsf{part}\ \{0,\ldots,\mathsf{brn}\ c-1\}\ P\ \wedge\ \mathsf{part}\ \{0,\ldots,\mathsf{brn}\ d-1\}\ Q\ \wedge\ [F : P \to Q \text{ bijection}] \wedge$
$(\forall I \in P.\ \mathsf{Wt}\ c\ s\ I = \mathsf{Wt}\ d\ t\ (F\ I) \wedge$
$\qquad (\forall i \in I.\ \forall j \in F\ I.\ \mathsf{eff}\ c\ s\ i \sim \mathsf{eff}\ d\ t\ j\ \wedge\ \theta\ (\mathsf{cont}\ c\ s\ i)\ (\mathsf{cont}\ d\ t\ j)))$

Thus, $\mathsf{match}^{\mathsf{c}}_{\mathsf{c}}\ \theta\ c\ d$ states that there exist partitions P and Q of the branches of c and d and a bijective correspondence $F : P \to Q$ so that, for any corresponding sets of branches I and $F\ I$:

– The cumulated weights are the same.

– For any pair (i,j) of branches in these sets, the effects are indistinguishable and the continuations are in θ.

Strong bisimilarity, $\approx_{\mathsf{s}}$, is defined coinductively as the largest (i.e., weakest) relation satisfying $\forall c,d.\ c \approx_{\mathsf{s}} d \iff \mathsf{match}^{\mathsf{c}}_{\mathsf{c}}\ (\approx_{\mathsf{s}})\ c\ d$, or, equivalently, the largest relation satisfying $\forall c,d.\ c \approx_{\mathsf{s}} d \Longrightarrow \mathsf{match}^{\mathsf{c}}_{\mathsf{c}}\ (\approx_{\mathsf{s}})\ c\ d$. If we ignore preservation of the state indistinguishability, this boils down to a well known property of Markov chains called probabilistic bisimulation or lumpability [13, 15].

According to the insight obtained in [23], a good during-execution noninterference candidate should be compositional with the language constructs and weaker than both siso and discr. It turns out that $\approx_{\mathsf{s}}$ has many of these characteristics, in particular, it will be shown to commute with all the constructs except for While and ParT.

To compensate for the lack of ParT-compositionality of $\approx_{\mathsf{s}}$, we introduce a weaker relation, $\approx_{01}$, that we call 01-*bisimilarity* because it requires a step to be matched by either no step (a stutter move) or one step. Its characteristic matcher is

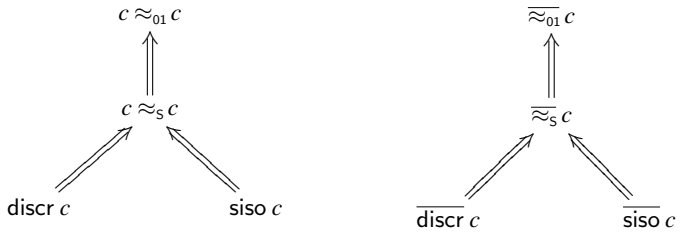

Fig. 3: Resumption-Based Notions and Syntactic Criteria

$$
\begin{array}{l}
\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ \theta\ c\ d \equiv \\
\forall s\ t.\ s \sim t \Longrightarrow \exists P\ Q\ I_0\ F. \\
\mathsf{part}\ \{0,\ldots,\mathsf{brn}\ c-1\}\ P\ \wedge\ \mathsf{part}\ \{0,\ldots,\mathsf{brn}\ d-1\}\ Q\ \wedge\ I_0 \in P\ \wedge\ [F : P \to Q \text{ bijection}] \\
(\forall I \in P - \{I_0\}.\ \dfrac{\mathsf{Wt}\ c\ s\ I}{1-\mathsf{Wt}\ c\ s\ I_0} = \dfrac{\mathsf{Wt}\ d\ t\ (F\ I)}{1-\mathsf{Wt}\ d\ t\ (F\ I_0)}\ \wedge \\
\qquad\qquad (\forall i \in I.\ \forall j \in F\ I.\ \mathsf{eff}\ c\ s\ i \sim \mathsf{eff}\ d\ t\ j\ \wedge\ \theta\ (\mathsf{cont}\ c\ s\ i)\ (\mathsf{cont}\ d\ t\ j))) \wedge \\
(\forall i \in I_0.\ s \sim \mathsf{eff}\ c\ s\ i\ \wedge\ \theta\ (\mathsf{cont}\ c\ s\ i)\ d)\ \wedge \\
(\forall j \in F\ I_0.\ t \sim \mathsf{eff}\ d\ t\ j\ \wedge\ \theta\ c\ (\mathsf{cont}\ d\ t\ j))
\end{array}
$$

$\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ \theta\ c\ d$ relaxes $\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}$ to allow matching continuation steps not only by continuation steps, but also by stutter moves. Thus, in the partitions P and Q, one singles out sets of stutter branches I_0 and $F\ I_0$ whose effects are required to preserve indistinguishability and whose continuations are required to be in relation θ with the other party's source command, d or c. Moreover, the cumulated weights in corresponding non-stutter sets of branches are no longer required to be equal, but only equal relatively to the cumulated weights of all non-stutter branches (i.e., to 1 minus the cumulated weights of the stutter ones).

01-*bisimilarity*, $\approx_{01}$, is now defined analogously to $\approx_{\mathsf{s}}$, as the largest relation satisfying $\forall c\ d.\ c \approx_{01} d \iff \mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ (\approx_{01})\ c\ d$. The notion of security associated to a bisimilarity is its diagonal version, which we call self bisimilarity. Thus, c is called *self strongly-bisimilar* if $c \approx_{\mathsf{s}} c$ and *self* 01-*bisimilar* if $c \approx_{01} c$.

The 01-steps relaxation scheme is well-known in language-based possibilistic bisimulations [4–6, 23], and corresponds to the triangle unwinding scheme in system-based security [16]—our relation $\approx_{01}$ seems to be its first probabilistic adaptation.

Note that all the above four notions of security are defined employing universal quantification over the relevant current states (either s alone for discr or the indistinguishable states s and t for the other notions), which are therefore "refreshed" at each resumption point. This means that security is defined *interactively*, guaranteeing correct behavior under the assumption that the environment (consisting perhaps of the attacker or of the other threads) may change the state at any point. This is crucial for compositionality [23, 24, 26]. It is immediate to prove by coinduction that $\approx_{01}$ is weaker than $\approx_{\mathsf{s}}$, which in turn is weaker than siso and discr:

Prop 1 *The implications shown in the left of Fig. 3 hold.*

3.2 Compositionality

Here we establish the compositionality properties of the two resumption-based notions w.r.t. the language constructs and discuss their relative strengths and weaknesses.

First, we need atomic properties of preservation and compatibility adapted to the abstract notions of state and indistinguishability relation. An atom *atm* is called $\sim$*-preserving*, written $\mathsf{pres}\ atm$, if $\forall s.\ \mathsf{aexec}\ atm\ s \sim s$; it is called $\sim$*-compatible*, written $\mathsf{cpt}\ atm$, if $\forall s\ t.\ s \sim t \Longrightarrow \mathsf{aexec}\ atm\ s \sim \mathsf{aexec}\ atm\ t$. A test *tst* is called $\sim$*-compatible*, written $\mathsf{cpt}\ tst$, if $\forall s\ t.\ s \sim t \Longrightarrow \mathsf{tval}\ tst\ s = \mathsf{tval}\ tst\ t$. A choice *ch* is called $\sim$*-compatible*, written $\mathsf{cpt}\ ch$, if $\forall s\ t.\ s \sim t \Longrightarrow \mathsf{cval}\ ch\ s = \mathsf{cval}\ ch\ t$.

In the setting of Example 2, for atoms, $\sim$-preservation means no assignment to low variables and $\sim$-compatibility means no direct leaks, i.e., no assignment to low variables of expressions depending on high variables. Moreover, for tests, $\sim$-compatibility means no dependence on high variables. A stateless choice is always compatible and an "if" choice is compatible if it is so as a test.

The next proposition states various compositionality results, schematically represented in Fig. 4 as follows. The first column lists the possible forms of a command c (c may be an atom *atm*, or have the form $\mathsf{Seq}\ c_1\ c_2$, etc.). The next columns list conditions under which the predicates stated on the first row hold for c. Thus, e.g., row 4 column 3 says: if $\mathsf{cpt}\ ch$, $\mathsf{siso}\ c_1$ and $\mathsf{siso}\ c_2$, then $\mathsf{siso}\ (\mathsf{Ch}\ ch\ c_1\ c_2)$. The horizontal line in row 3 columns 4 and 5 represents an "or"—thus, e.g., row 3 column 4 says: if either [$\mathsf{siso}\ c_1$ and $c_2 \approx_{\mathsf{s}} c_2$] or [$c_1 \approx_{\mathsf{s}} c_1$ and $\mathsf{discr}\ c_2$] then $\mathsf{Seq}\ c_1\ c_2 \approx_{\mathsf{s}} \mathsf{Seq}\ c_1\ c_2$.

Prop 2 *The compositionality facts stated in Fig. 4 hold.*

As expected, compatibility is a minimal security requirement for atoms, with discreetness requiring even preservation. Sequential composition behaves perfectly w.r.t. siso and discr, but for $\approx_{\mathsf{s}}$ and $\approx_{01}$ it requires strengthening either to siso on the left component or to discr on the right component. For choice, compatibility is required for all notions except for discr; in the particular case of "if" tests, this becomes the well-known "no high test" condition; for stateless choices, the condition is vacuously true.

The compositionality w.r.t. Par and ParT reveal some interesting phenomena. While in the possibilistic case we have shown that, in the presence of the aforementioned interactivity proviso, parallel composition is unconditionally compositional, here the situation is less convenient. Possibilistically, it makes no difference whether or not a finished thread is removed from the pool. Indeed, scheduling it to take a stutter move has no possibilistic effect. However, it does have the effect of delaying the steps taken by other threads. This is already discussed in [28, 29] for sequential threads, and is reflected in our case by $\approx_{\mathsf{s}}$ not being compositional w.r.t. ParT. Also, $\approx_{01}$ is not ParT-compositional either. Fortunately, ParT composition of $\approx_{\mathsf{s}}$-related threads yields $\approx_{01}$-related results, which saves the day—this is the main reason for introducing $\approx_{01}$.

Another problem that seems specific to probabilistic semantics is that $\approx_{\mathsf{s}}$ and $\approx_{01}$ are not compositional w.r.t. While. The main reason is that both $\approx_{\mathsf{s}}$ and $\approx_{01}$ are termination-insensitive, and hence they do not detect, in a while loop, when the body command has finished executing, which makes synchronization impossible in the bisimilarity game.

These compositionality problems are actually not as bad as they may seem: as we discuss in §3.3, when proving noninterference of a command c, if a notion fails to be compositional w.r.t. the construct from the top of c, one can fall back on a stronger notion, for which the proof can progress.

c	$\mathsf{discr}\ c$	$\mathsf{siso}\ c$	$c \approx_{\mathsf{s}} c$	$c \approx_{01} c$
atm	$\mathsf{pres}\ atm$	$\mathsf{cpt}\ atm$	$\mathsf{cpt}\ atm$	$\mathsf{cpt}\ atm$
$\mathsf{Seq}\ c_1\ c_2$	$\mathsf{discr}\ c_1$ $\mathsf{discr}\ c_2$	$\mathsf{siso}\ c_1$ $\mathsf{siso}\ c_2$	$\mathsf{siso}\ c_1$ $c_2 \approx_{\mathsf{s}} c_2$ $c_1 \approx_{\mathsf{s}} c_1$ $\mathsf{discr}\ c_2$	$\mathsf{siso}\ c_1$ $c_2 \approx_{01} c_2$ $c_1 \approx_{01} c_1$ $\mathsf{discr}\ c_2$
$\mathsf{Ch}\ ch\ c_1\ c_2$	$\mathsf{discr}\ c_1$ $\mathsf{discr}\ c_2$	$\mathsf{cpt}\ ch$ $\mathsf{siso}\ c_1$ $\mathsf{siso}\ c_2$	$\mathsf{cpt}\ ch$ $c_1 \approx_{\mathsf{s}} c_1$ $c_2 \approx_{\mathsf{s}} c_2$	$\mathsf{cpt}\ ch$ $c_1 \approx_{01} c_1$ $c_2 \approx_{01} c_2$
$\mathsf{While}\ tst\ d$	$\mathsf{discr}\ d$	$\mathsf{cpt}\ tst$ $\mathsf{siso}\ d$	False	False
$\mathsf{Par}\ [c_0,\ldots,c_{n-1}]$	$\mathsf{discr}\ c_l$ $0 \le l < n$	$\mathsf{siso}\ c_l$ $0 \le l < n$	$c_l \approx_{\mathsf{s}} c_l$ $0 \le l < n$	False
$\mathsf{ParT}\ [c_0,\ldots,c_{n-1}]$	$\mathsf{discr}\ c_l$ $0 \le l < n$	False	False	$c_l \approx_{\mathsf{s}} c_l$ $0 \le l < n$

Fig. 4: Compositionality of Resumption-Based Noninterference

3.3 Syntactic Criteria

With the implications between bisimilarities and their compositionality facts in place, we can automatically infer type-system criteria. This was described in [23, §6] as a "table-and-graph" method for a possibilistic programming language. Since the analysis from there is semantics-independent, it also applies here. For each security notion $\chi \in \{\mathsf{discr}, \mathsf{siso}, \approx_{\mathsf{s}}, \approx_{01}\}$, we define a function $\overline{\chi} : \mathbf{com} \to \mathbf{bool}$ following a potential attempt to prove $\chi\ c$, first using the corresponding compositionality fact from Fig. 4's table, and, if this fails, falling back on stronger notions given by the predecessors of χ from Fig. 3's left graph. Thus, the cell corresponding to the form of c and the notion χ contains some properties of the components of c and possibly a side condition. Then:
– if the side condition holds, then $\overline{\chi}\ c$ is defined recursively as the conjunction of all $\overline{\chi'}\ c'$, where $\chi'\ c'$ are the listed conditions for the components c' of c;
– otherwise, $\overline{\chi}\ c$ is defined as the disjunction of all $\overline{\chi'}\ c$, where χ' are the immediate predecessors of χ in Fig. 3's left graph.

Concretely:

$\overline{\mathsf{discr}}\ atm \iff \mathsf{pres}\ atm$; $\overline{\mathsf{discr}}\ (\mathsf{Seq}\ c_1\ c_2) \iff \overline{\mathsf{discr}}\ (\mathsf{Ch}\ ch\ c_1\ c_2) \iff \overline{\mathsf{discr}}\ c_1 \wedge \overline{\mathsf{discr}}\ c_2$;
$\overline{\mathsf{discr}}\ (\mathsf{While}\ tst\ d) \iff \overline{\mathsf{discr}}\ d$;
$\overline{\mathsf{discr}}\ (\mathsf{Par}\ [c_0,\ldots,c_{n-1}]) \iff \overline{\mathsf{discr}}\ (\mathsf{ParT}\ [c_0,\ldots,c_{n-1}]) \iff \bigwedge_{i=0}^{n-1} \overline{\mathsf{discr}}\ c_i$;
$\overline{\mathsf{siso}}\ atm \iff \mathsf{cpt}\ atm$; $\overline{\mathsf{siso}}\ (\mathsf{Seq}\ c_1\ c_2) \iff \overline{\mathsf{siso}}\ c_1 \wedge \overline{\mathsf{siso}}\ c_2$;
$\overline{\mathsf{siso}}\ (\mathsf{Ch}\ ch\ c_1\ c_2) \iff \mathsf{cpt}\ ch \wedge \overline{\mathsf{siso}}\ c_1 \wedge \overline{\mathsf{siso}}\ c_2$; $\overline{\mathsf{siso}}\ (\mathsf{While}\ tst\ d) \iff \mathsf{cpt}\ tst \wedge \overline{\mathsf{siso}}\ d$;
$\overline{\mathsf{siso}}\ (\mathsf{Par}\ [c_0,\ldots,c_{n-1}]) \iff \bigwedge_{i=0}^{n-1} \overline{\mathsf{siso}}\ c_i$; $\overline{\mathsf{siso}}\ (\mathsf{ParT}\ [c_0,\ldots,c_{n-1}]) \iff \mathsf{False}$;
$\overline{\approx_{\mathsf{s}}}\ atm \iff \mathsf{cpt}\ atm$; $\overline{\approx_{\mathsf{s}}}\ (\mathsf{Seq}\ c_1\ c_2) \iff (\overline{\mathsf{siso}}\ c_1 \wedge \overline{\approx_{\mathsf{s}}}\ c_2) \vee (\overline{\approx_{\mathsf{s}}}\ c_1 \wedge \overline{\mathsf{discr}}\ c_2)$;
$\overline{\approx_{\mathsf{s}}}\ (\mathsf{Ch}\ ch\ c_1\ c_2) \iff \mathsf{cpt}\ ch \wedge \overline{\approx_{\mathsf{s}}}\ c_1 \wedge \overline{\approx_{\mathsf{s}}}\ c_2$;
$\overline{\approx_{\mathsf{s}}}\ (\mathsf{While}\ tst\ d) \iff \overline{\mathsf{siso}}\ (\mathsf{While}\ tst\ d) \vee \overline{\mathsf{discr}}\ (\mathsf{While}\ tst\ d)$;
$\overline{\approx_{\mathsf{s}}}\ (\mathsf{Par}\ [c_0,\ldots,c_{n-1}]) \iff \bigwedge_{i=0}^{n-1} \overline{\approx_{\mathsf{s}}}\ c_i$;
$\overline{\approx_{\mathsf{s}}}\ (\mathsf{ParT}\ [c_0,\ldots,c_{n-1}]) \iff \overline{\mathsf{siso}}\ (\mathsf{ParT}\ [c_0,\ldots,c_{n-1}]) \vee \overline{\mathsf{discr}}\ (\mathsf{ParT}\ [c_0,\ldots,c_{n-1}])$;
$\overline{\approx_{01}}\ atm \iff \mathsf{cpt}\ atm$; $\overline{\approx_{01}}\ (\mathsf{Seq}\ c_1\ c_2) \iff (\overline{\mathsf{siso}}\ c_1 \wedge \overline{\approx_{01}}\ c_2) \vee (\overline{\approx_{01}}\ c_1 \wedge \overline{\mathsf{discr}}\ c_2)$;

$\overline{\approx_{01}}\,(\mathsf{Ch}\,ch\,c_1\,c_2) \iff \mathsf{cpt}\,ch \wedge \overline{\approx_{01}}\,c_1 \wedge \overline{\approx_{01}}\,c_2$; $\overline{\approx_{01}}\,(\mathsf{While}\,tst\,d) \iff \overline{\approx_{s}}\,(\mathsf{While}\,tst\,d)$;
$\overline{\approx_{01}}\,(\mathsf{Par}\,[c_0,\ldots,c_{n-1}]) \iff \overline{\approx_{s}}\,(\mathsf{Par}\,[c_0,\ldots,c_{n-1}])$;
$\overline{\approx_{01}}\,(\mathsf{ParT}\,[c_0,\ldots,c_{n-1}]) \iff \bigwedge_{i=0}^{n-1} \overline{\approx_{s}}\,c_i$.

The above are valid recursive definitions: each operator $\overline{\chi}$ is defined recursively in terms of itself and/or in terms of previously defined operators. Note how, when compositionality for a notion fails, stronger notions are invoked, e.g.:
$\overline{\approx_{s}}\,(\mathsf{While}\,tst\,d) \iff \overline{\mathsf{siso}}\,(\mathsf{While}\,tst\,d) \vee \overline{\mathsf{discr}}\,(\mathsf{While}\,tst\,d) \iff (\mathsf{cpt}\,tst \wedge \overline{\mathsf{siso}}\,d) \vee \overline{\mathsf{discr}}\,d$

We can prove that the syntactic notions are indeed sufficient criteria for their semantic counterparts, and that the former inherit the hierarchy of the latter.

Prop 3 *(1) For any χ in Fig. 3 left, it holds that $\forall c.\ \overline{\chi}\ c \Longrightarrow \chi\ c$.*
(2) The implications shown in the right of Fig. 4 hold.

The next example illustrates the semantic notions and their syntactic criteria:

Example 3 Consider the following commands, with h, h' high and l, l' low variables.
– d_0: $h' := 0$; while $h > 0$ do $\mathsf{Ch}_{0.5}\,(h := 0)\,(h' := h' + 1)$
– d_1: while $h > 0$ do $\mathsf{Ch}_{0.5}\,(h := h - 1)\,(h := h + 1)$
– d_2: if $l = 0$ then $l' := 1$ else d_0
– d_3: $h := 5$; $(d_0 \parallel_T l := 1)$
– d_4: $(\text{if } h = 0 \text{ then } h := 1 \text{ ; } h := 2 \text{ else } h := 3)$; $l := 4$
– d_5: $d_4 \parallel_T l := 5$

Provided initially $h > 0$, d_0 has the effect of assigning a random geometrically distributed integer value to h', and d_1 performs a one-dimensional random walk with the value of h (the so-called gambler's ruin), resulting invariably in exit (at $h = 0$) with probability 1. d_3 illustrates one advantage of being able to nest parallel composition inside sequential composition—the possibility to make some global initializations (here, $h := 5$) before starting up the thread pool (here, containing two threads, d_0 and $l := 1$).

d_0 and d_1 are discreet, d_2 is self strongly bisimilar, and d_3 is self 01-bisimilar, but not self strongly bisimilar due to the presence of ParT whose compositionality requires the shift from $\approx_s$ to $\approx_{01}$. (d_3 would become self strongly bisimilar had we replaced ParT with Par.) Indeed, d_0–d_3 are deemed secure by the syntactic criteria from Fig. 3, e.g.:

$$
\begin{array}{lllllllll}
 & & & & \overline{\approx_{01}}\,d_3 & & & & \iff \\
\overline{\mathsf{siso}}\,(h := 5) & \wedge & \overline{\approx_{01}}\,(d_0 \parallel_T l := 1) & & \vee & & \overline{\approx_{01}}\,(h := 5) & \wedge & \overline{\mathsf{discr}}\,(d_0 \parallel_T l := 1) \iff \\
\mathsf{True} & \wedge & \overline{\approx_{s}}\,d_0 \wedge \overline{\approx_{s}}\,(l := 1) & & \vee & & \mathsf{True} & \wedge & \overline{\mathsf{discr}}\,d_0 \wedge \overline{\mathsf{discr}}\,(l := 1) \iff \\
 & & & & \ldots & & & & \iff \\
\mathsf{True} & \wedge & \mathsf{True} \wedge \mathsf{True} & & \vee & & \mathsf{True} & \wedge & \mathsf{True} \wedge \mathsf{False} \iff \\
 & & \mathsf{True} & & \vee & & \mathsf{False} & & \iff \\
 & & & & \mathsf{True} & & & &
\end{array}
$$

On the other hand, d_4 and d_5 are not secure, not even according to $\approx_{01}$. The problem with d_4 is that the timing of the low assignment $l := 4$ depends on the value of the high variable h, which can cause probabilistic leaks when placed in parallel with other threads that may update l. d_5 shows such a situation: the initial value of h influences the likelihood that $l := 4$ is executed before $l := 5$. And indeed, d_4 and d_5 are rejected by all the syntactic criteria, e.g.:

$$
\begin{array}{lllllllll}
 & & & & \overline{\approx_{01}}\,d_4 & & & & \iff \\
\overline{\mathsf{siso}}\,(\text{if } h = 0 \ldots) & \wedge & \overline{\approx_{01}}\,(l := 4) & & \vee & & \overline{\approx_{01}}\,(\text{if } h = 0 \ldots) & \wedge & \overline{\mathsf{discr}}\,(l := 4) \iff \\
 & & & & \ldots & & & & \iff \\
\mathsf{False} & \wedge & \mathsf{True} & & \vee & & \mathsf{True} & \wedge & \mathsf{False} \iff \\
 & & & & \mathsf{False} & & & &
\end{array}
$$

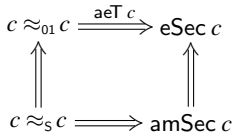

Fig. 5: Resumption-Based and Trace-Based Notions of Security

3.4 Trace-Based Noninterference

Both $\approx_{\mathsf{s}}$ and $\approx_{01}$ protect against the following end-to-end kind of probabilistic attack: The attacker may run the program multiple times and collect statistical information about the distribution of the final state up to $\sim$ (which corresponds to the low part of the memory); however, this data will never allow the attacker to infer anything about the initial state beyond the $\sim$-abstraction (which corresponds to the high part of the memory). Such a property is best formalized as trace-based noninterference.

For technical reasons, all our execution traces are infinite, with dummy transitions added for finished commands. We call *terminating* those traces reaching a configuration whose command is finished: $\mathsf{termin}\ (c_i,s_i)_{i\in\mathbf{nat}} \equiv \exists i \in \mathbf{nat}.\ \mathsf{finished}\ c_i$. Since dummy transitions do not affect the state, *the final state* of a terminating trace, $\mathsf{fstate}\ (c_i,s_i)_{i\in\mathbf{nat}}$, is well defined as the unique s such that $\exists i.\ s = s_i \wedge \mathsf{finished}\ c_i$.

We define the following sets of traces, for any (c,s), n and t:
$T_{(c,s),n,t} \equiv \{(c_i,s_i)_{i\in\mathbf{nat}} \in \mathsf{Trace}_{(c,s)} \mid s_n \sim t\}$,
the set of (c,s)-traces whose n-th configuration's state is indistinguishable from t;
$T_{(c,s),t} \equiv \{(c_i,s_i)_{i\in\mathbf{nat}} \in \mathsf{Trace}_{(c,s)} \mid \mathsf{termin}\ (c_i,s_i)_{i\in\mathbf{nat}} \wedge \mathsf{fstate}\ (c_i,s_i)_{i\in\mathbf{nat}} \sim t\}$,
the set of terminating (c,s)-traces whose final state is indistinguishable from t.

The set of terminating states, as well as $T_{(c,s),n,t}$ and $T_{(c,s),t}$, are all measurable sets since they can be written as countable unions of countable intersections of basic events. We say c *almost everywhere terminates*, written $\mathsf{aeT}\ c$, if $\forall s.\ \Pr_{(c,s)}\ \{(c_i,s_i)_{i\in\mathbf{nat}} \in \mathsf{Trace}_{(c,s)} \mid \mathsf{termin}\ (c_i,s_i)_{i\in\mathbf{nat}}\} = 1$, i.e., the set of terminating (c,s)-traces has measure 1.

We can now define the following trace-based notions of noninterference:
– *Any-moment security* states that, for any two executions starting in indistinguishable states and any given time, the probability of being at that time in any given indistinguishability class is the same:

$$\mathsf{amSec}\ c \equiv \forall s_1\ s_2.\ s_1 \sim s_2 \Longrightarrow \forall n\, t.\ \Pr_{(c,s_1)}\ T_{(c,s_1),n,t} = \Pr_{(c,s_2)}\ T_{(c,s_2),n,t}$$

– *End security* states that, for any two executions starting in indistinguishable states, the probability of ending up in any given indistinguishability class is the same:

$$\mathsf{eSec}\ c \equiv \forall s_1\ s_2.\ s_1 \sim s_2 \Longrightarrow \forall t.\ \Pr_{(c,s_1)}\ T_{(c,s_1),t} = \Pr_{(c,s_2)}\ T_{(c,s_2),t}$$

Any-moment security is a strong guarantee: even if one is able to observe the distribution of the low memory at any given moment, one still cannot infer anything about the initial high memory. On the other hand, end security warrants something weaker: that the final distribution of the low memory tells nothing about the initial high memory. One can prove that $\approx_{\mathsf{s}}$ implies any-moment security, and that this in turn implies end security. More interestingly, $\approx_{01}$ implies end security if we also assume almost-everywhere termination; roughly, the last assumption is necessary to make sure that the "bisimulation noise" caused by stutter moves cannot delay synchronization forever, but eventually becomes negligible.

Prop 4 *The implications listed in Fig. 5 hold.*

In Example 3, d_0–d_3 are all $\approx_{01}$-secure programs and are also almost-everywhere terminating, hence they satisfy eSec by Prop. 4. Moreover, since d_2 is $\approx_S$-secure, it satisfies the stronger property amSec by Prop. 4. d_5 does not satisfy eSec, since the distribution of the final low memory reveals whether h is 0 or not: if $h = 0$, then 1 out of 4 executions yields $l = 4$; otherwise, only 1 out of 8. On the other hand, even though d_4 is not $\approx_{01}$-secure, it obviously satisfies eSec, since all its executions yield $l = 4$.

4 Overview and Statistics

Our formal development [21] amounts to about 8000 lines of scripts in Isabelle [19]. Fig. 6 shows the main theory structure, indicating for each theory the number of lines and the corresponding sections of the paper. The types and functions parameterizing the language syntax and semantics, as well as the state-indistinguishability relation $\sim$, are fixed in Isabelle locales [12]—theory Concrete instantiates these locales as discussed in Examples 1 and 2.

The language semantics was tedious to formalize due to parallel composition (especially the termination-sensitive one), which involves list-index manipulation—employing iterated binary parallel composition instead (as in the possibilistic case [23]) was not an option, since the scheduler needs to address the thread pool as a whole. On the other hand, probabilistic semantics displays a certain conceptual simplification over traditional nondeterministic semantics: for each language construct, the direct rules and the inversion rules are merged into "direct" quantitative equations (as described in Fig. 1).

We defined the probabilistic bisimulations on the concrete branches of the operational semantics, and not on the more abstract Markov-chain transitions (which may identify some of the branches); indeed, the branches provided us with a good notation for partitioning the continuations and, more importantly, with the right level of abstraction for proving compositionality facts without having to query whether some continuations happen to be equal. On the other hand, we used the general-purpose Markov-chain construction of the traces and their probability space [9] as opposed to building them from branches, which of course saved us much background work but complicated a little the proofs relating resumption notions with trace notions.

The largest and most laborious part (roughly 36% of the whole development) deals with the compositionality results listed in Prop. 2—the bisimulation relations provided as witnesses in coinductive proofs involved tedious constructions of partitions and sums over sets. Isabelle's Sledgehammer tool for deploying external automatic theorem provers [20], very helpful in discharging goals on possibilistic bisimulations, was less helpful here, where the $\forall\exists$ scheme of traditional bisimulations gives way to a quantitative $\forall\sum$ scheme. Another laborious task was establishing the connection between trace-based and resumption-based notions, which also involved heavy sum reasoning.

Having the compositionality preparations, the inference of syntactic criteria (Prop. 3) was immediate, with the induction goals discharged automatically by the simplifier and the classical reasoner. The route through compositionality facts localized at each language construct does better justice to noninterference results than previous formulations from the literature [17, 27–29, 31], which rely on complex monolithic proofs.

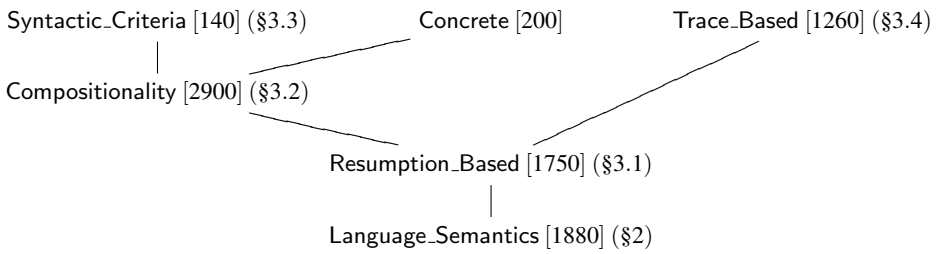

Fig. 6: Isabelle Theory Structure

5 Conclusions and Related Work

We have formalized noninterference properties for a multi-threaded language with probabilistic choice and uniform scheduler. Distinguishing features of our approach are the comprehensive study, covering both resumption-based and trace-based notions, and the automatic extraction of syntactic criteria from compositionality. Moreover, all previous work[2] considers systems of sequential deterministic threads run in parallel by a probabilistic scheduler. Our language is more powerful, allowing pervasive probabilistic behavior, including probabilistic threads. This makes the mathematical analysis more challenging, since each thread yields a Markov chain, which needs to be combined by parallel composition in the larger Markov chain of the thread pool. In fact, the very notions of thread and thread pool are relative here, since the language allows nesting parallel composition into other constructs (e.g., having Seq on top of Par or ParT), although, as we have seen, security requirements restrict some of this expressiveness.

If we are to identify a "pen-and-paper" reference for our formalized resumption-based notions, the closest is a series of papers by Smith and others [27–29, 31], which progressively introduces notions analogous to ours. Specifically, [31] introduces self isomorphism, [27] strong bisimilarity, and [28,29] a notion weaker than our 01-bisimilarity called weak bisimilarity. In each case, the type system proved sound in there is equivalent to our syntactic criterion uniformly extracted from compositionality (Prop. 3).

A further point of convergence with the above works is the consideration of various flavors of parallel composition. [31] and [27] consider the termination-insensitive Par, while later work [28, 29] focuses on the termination-sensitive ParT. Retrospectively, in the light of the compositionality facts of Prop. 2, this is not surprising, since siso and $\approx_{\mathsf{S}}$ are both compositional w.r.t. Par, and $\approx_{01}$ is quasi-compositional w.r.t. ParT.

Interestingly, all of the above works prove that the type system implies the corresponding resumption-based version, but they allude informally to a trace-based notion as the ultimately targeted security guarantee. E.g., [27, page 10] reads: "the probability that the low variables have certain values after k steps is the same when starting from (O,μ) as where starting from (O,ν)" (where (O,μ) and (O,ν) are bisimilar configurations)—we have formalized this as any-moment security. Also, [28, page 8] reads: "the probability that the low variables end up with some values from (O,μ)

[2] We only discuss the most related work, covering probabilistic languages and noninterference. A survey of related work on possibilistic noninterference is given in [23].

is the same as the probability that they end up with those values from (O, v)"—we have formalized this as end security. Establishing formally the relationship between a resumption-based notion and a trace-based notion can range from routine (as in $\approx_S$ versus any-moment security) to highly nontrivial (as in $\approx_{01}$ versus end security).

There are some extensions and generalizations of probabilistic semantics and noninterference not covered by our formalization. Smith [28] also considers a protect command enforcing atomicity of execution. (In principle, our formalization can handle this language construct by "instantiating" the **atom** parameter to a type mutually recursive with **com**.) He also sketches an extension to dynamic thread creation. Sabelfeld and Sands [26] show that a type system corresponding to our syntactic criterion $\overline{\mathsf{siso}}$ for self isomorphism is strong enough to ensure noninterference for any scheduler, not only the uniform one. Mantel and Sudbrock [17] relax the $\overline{\mathsf{siso}}$ requirement, while still covering relevant schedulers such as uniform and round robin. Our own "pen-and-paper" work [22] generalizes the results of Smith based on weak bisimilarity [28, 29] to a different class of schedulers than Mantel and Sudbrock's, providing an arguably more manageable criterion for schedulers and a stronger security guarantee.

Our end security is a generalization of the one from [17], where one defines the property only for globally terminating thread pools—this simplifying assumption allows an elementary treatment of the relevant probabilities, not requiring measure theory. We relax the termination requirement to almost-everywhere termination. This relaxation is relevant for probabilistic languages, where many interesting programs terminate only almost everywhere—this also happens to be the case for d_0–d_3 in Example 3.

Probabilistic, but single-threaded languages in the style of pGCL [18] have been formalized before in HOL4 [11], Coq [2] and Isabelle [8]. In very recent work [7], Cock verifies in Isabelle a lattice scheduler (a uniform scheduler that distinguishes between high and low processes) aimed at closing covert channels such as cash channels. The scheduler is shown compatible with the possibilistic refinement framework underlying the verification of the seL4 operating system kernel [14]. While the work does not target a programming language, the scheduler itself is specified as a program in pGCL and shown probabilistically noninterfering w.r.t. a version of lumpability.

Acknowledgments. This work was supported by the DFG project Ni 491/13–2, part of the DFG priority program RS3 and the DFG RTG 1480.

References

1. The POPLmark challenge, 2009. http://www.seas.upenn.edu/ plclub/poplmark/.
2. P. Audebaud and C. Paulin-Mohring. Proofs of randomized algorithms in Coq. *S. of Comp. Prog.*, 74(8):568–589, 2009.
3. G. Barthe, P. R. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *CSFW*, pages 100–114, 2004.
4. G. Barthe and L. P. Nieto. Formally verifying information flow type systems for concurrent and thread systems. In *FMSE*, pages 13–22, 2004.
5. G. Boudol. On typing information flow. In *ICTAC*, pages 366–380, 2005.
6. G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1-2):109–130, 2002.
7. D. Cock. Practical probability: Applying pGCL to lattice scheduling. ITP 2013, to appear.

8. D. Cock. Verifying probabilistic correctness in Isabelle with pGCL. In *SSV*, pages 167–178, 2012.
9. J. Hölzl. Analyzing discrete-time Markov chains with countable state space in Isabelle/HOL. Submitted to CPP 2013.
10. J. Hölzl and T. Nipkow. Verifying pCTL model checking. In *TACAS*, pages 347–361, 2012.
11. J. Hurd, A. McIver, and C. Morgan. Probabilistic guarded commands mechanized in HOL. *Theor. Comput. Sci.*, 346(1), 2005.
12. F. Kammüller, M. Wenzel, and L. C. Paulson. Locales - a sectioning concept for Isabelle. In *TPHOLs'99*, pages 149–166, 1999.
13. J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov chains (second edition)*. Springer, 1976.
14. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. sel4: formal verification of an os kernel. In *SOSP*, pages 207–220, 2009.
15. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1 – 28, 1991.
16. H. Mantel. A uniform framework for the specification and verification of security properties. Ph.D. thesis, Univ. of Saarbrücken, 2003.
17. H. Mantel and H. Sudbrock. Flexible scheduler-independent security. In *ESORICS*, pages 116–133, 2010.
18. A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.
19. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer, 2002.
20. L. C. Paulson and J. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In *IWIL*, 2010.
21. A. Popescu and J. Hölzl. Formal development associated with this paper. `http://www21.in.tum.de/~popescua/prob.zip`.
22. A. Popescu, J. Hölzl, and T. Nipkow. Noninterfering schedulers. To be presented at CALCO 2013. `http://www21.in.tum.de/~popescua/pdf/CALCO2013.pdf`.
23. A. Popescu, J. Hölzl, and T. Nipkow. Proving concurrent noninterference. In *CPP*, pages 109–125, 2012.
24. A. Sabelfeld. Confidentiality for multithreaded programs via bisimulation. In *International Conference on Perspectives of System Informatics*, LNCS, pages 260–273, 2003.
25. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
26. A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *CSFW*, pages 200–214, 2000.
27. G. Smith. A new type system for secure information flow. In *CSFW*, pages 115–125, 2001.
28. G. Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *CSFW*, pages 3–13, 2003.
29. G. Smith. Improved typings for probabilistic noninterference in a multi-threaded language. *Journal of Computer Security*, 14(6):591–623, 2006.
30. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *POPL*, pages 355–364, 1998.
31. D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *Journal of Computer Security*, 7(2,3):231–253, 1999.
32. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2,3):167–187, 1996.
33. S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *CSFW*, pages 29–43, 2003.

Appendix: Proof Ideas

This appendix is included for the reviewers' convenience and is not required for understanding the content of the main paper.

Proof of Prop. 2. For statements involving $\approx_{\mathsf{s}}$ and $\approx_{01}$, we prove the more general form that does not assume the arguments equal. All proofs are by coinduction, exhibiting a suitable post-fixpoint relation that implies the conclusion.

We sketch the proof of a representative case, in row 7 column 5: $(\bigwedge_{k=0}^{n-1} c_k \approx_{\mathsf{s}} d_k) \Longrightarrow \mathsf{ParT}\,[c_0,\ldots,c_{n-1}] \approx_{01} \mathsf{ParT}\,[d_0,\ldots,d_{n-1}]$. To prove this by coinduction on the definition of $\approx_{01}$, we let $\theta\ c\ d \equiv \exists (c_k,d_k)_{k=0}^{n-1}.\ (\bigwedge_{i=0}^{n-1} c_k \approx_{\mathsf{s}} d_k) \wedge c = \mathsf{ParT}\,[c_0,\ldots,c_{n-1}] \wedge d = \mathsf{ParT}\,[d_0,\ldots,d_{n-1}]$ and assume $\theta\ c\ d$; we need to show $\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ \theta\ c\ d$.

For each $k \in \{0,\ldots,n-1\}$, from $c_k \approx_{\mathsf{s}} d_k$ we have $\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}\ (\approx_{\mathsf{s}})\ c_k\ d_k$, hence we obtain the partitions P_k of $\mathsf{brn}\ c_k$ and Q_k of $\mathsf{brn}\ d_k$ and the function $F_k : P_k \to Q_k$ as in the definition of $\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}$. From these, we construct suitable partitions P of $\mathsf{brn}\ c$ and Q of $\mathsf{brn}\ d$ and set $I_0 \in P$ of stutter-move indexes as in the definition of $\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}$ as follows. Let G_k be the shifting of P_k with the branches of the previous commands in the list, $\{\{\sum_{l=0}^{k-1} \mathsf{brn}\ c_l + i \mid i \in I\} \mid I \in P_k\}$, and, similarly, $H_k = \{\{\sum_{l=0}^{k-1} \mathsf{brn}\ d_l + j \mid j \in J\} \mid J \in Q_k\}$. We let $P = \bigcup_{k=0}^{n-1} P_k$, $Q = \bigcup_{k=0}^{n-1} Q_k$ and define $F : P \to Q$ by $F\ G_k = H_k$. We let $I_0 = \bigcup\{G_k \mid 0 < k < n \wedge (\mathsf{finished}\ c_k \vee \mathsf{finished}\ d_k)\}$—intuitively, whenever one of the threads of the pool is finished, it and its matching thread are marked as stutter; this is justified by the fact that any thread strongly bisimilar to a finished thread is discreet. The conditions from the definition of $\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}$ can now be checked using the operational semantics of ParT—this involves some straightforward but tedious computation with sums. □

Proof of Prop. 3. Both points follow by structural induction on c. For (1), the proof is straightforward, since the defining clauses of $\overline{\chi}$ were chosen to be properties of χ. □

Proof of Prop. 4.
$c \approx_{\mathsf{s}} c \Longrightarrow c \approx_{01} c$ is already covered by Prop. 1.
$\mathsf{amSec}\ c \Longrightarrow \mathsf{eSec}\ c$: Let $F_{(c,s),n,t}$ be the set of all (c,s)-traces whose command becomes finished precisely at moment n with final state indistinguishable from t, namely, $\{(c_i,s_i)_{i\in\mathbf{nat}} \in \mathsf{Trace}_{(c,s)} \mid \mathsf{finished}\ c_n \wedge \forall i < n.\ \neg\,\mathsf{finished}\ c_i\}$. From $\mathsf{amSec}\ c$, we have that, given $s_1 \sim s_2$, $\Pr_{(c_1,s_1)}\ F_{(c_1,s_1),n,t} = \Pr_{(c_2,s_2)}\ F_{(c_2,s_2),n,t}$. Now $\mathsf{eSec}\ c$ follows from $T_{(c,s),t} = \bigcup_{n\in\mathbf{nat}} F_{(c,s),n,t}$ and the sets $F_{(c,s),n,t}$ being disjoint.
To prove the remaining implications, we use the following properties of $T_{(c,s),n,t}$ and $T_{(c,s),t}$, where $\mathsf{next}\ c\ s\ i$ denotes $(\mathsf{cont}\ c\ s\ i, \mathsf{eff}\ c\ s\ i)$:

$$\Pr_{(c,s)}\ T_{(c,s),n+1,t} = \textstyle\sum_{i\in\mathsf{brn}\ c} \mathsf{wt}\ c\ s\ i \ast \Pr_{\mathsf{next}\ c\ s\ i}\ T_{\mathsf{next}\ c\ s\ i,n,t} \qquad (*)$$

$$\Pr_{(c,s)}\ T_{(c,s),t} = \textstyle\sum_{i\in\mathsf{brn}\ c} \mathsf{wt}\ c\ s\ i \ast \Pr_{\mathsf{next}\ c\ s\ i}\ T_{\mathsf{next}\ c\ s\ i,t} \qquad (**)$$

$c \approx_{\mathsf{s}} c \Longrightarrow \mathsf{amSec}\ c$: We prove the slightly more general fact $c_1 \approx_{\mathsf{s}} c_2 \wedge s_1 \sim s_2 \Longrightarrow \Pr_{(c_1,s_1)}\ T_{(c_1,s_1),n,t} = \Pr_{(c_2,s_2)}\ T_{(c_2,s_2),n,t}$ by induction on n using the definition of $\approx_{\mathsf{s}}$'s matcher $\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}$ and (*).

$c \approx_{01} c \Longrightarrow \mathsf{eSec}\ c$: Let $N_{(c,s),n}$ be the set of traces whose $(n+1)$'th state is not discreet (which also means, by the definition of discreetness, that its previous states are not discreet either).

We first assume $c_1 \approx_{\mathsf{s}} c_2$ and show that

$$\mathsf{dist}\,(\Pr_{(c_1,s_1)}\ T_{(c_1,s_1),t},\ \Pr_{(c_2,s_2)}\ T_{(c_2,s_2),t}) \leq \Pr_{(c_1,s_1)}\ N_{(c_1,s_1),n} + \Pr_{(c_2,s_2)}\ N_{(c_2,s_2),m} \qquad (***)$$

(where dist is the standard distance between two reals) by induction on $n+m$ using the definition of $\approx_{01}$'s matcher $\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}$ and (**).

Now, we assume $c \approx_{01} c$ and $\mathsf{aeT}\ c$. To prove $\mathsf{eSec}\ c$, we assume $s_1 \sim s_2$. Since all finished commands are also discreet, from $\mathsf{eSec}\ c$ it follows that $\bigcap_{n \in \mathbf{nat}} \Pr_{(c,s_1)}\ N_{(c,s_1),n} = \bigcap_{n \in \mathbf{nat}} \Pr_{(c,s_2)}\ N_{(c,s_2),n} = \emptyset$, hence $\lim_{n\to\infty} \Pr_{(c,s_1)}\ N_{(c,s_1),n} = \lim_{n\to\infty} \Pr_{(c,s_2)}\ N_{(c,s_2),n} = 0$. With (***), it follows that $\Pr_{(c,s_1)}\ T_{(c,s_1),t} = \Pr_{(c,s_2)}\ T_{(c,s_2),t}$, as desired. □