Proving Concurrent Noninterference*

Andrei Popescu[1,2] Johannes Hölzl[1] Tobias Nipkow[1]

1: Technische Universität München 2: Institute of Mathematics Simion Stoilow, Romania

Abstract. We perform a formal analysis of compositionality techniques for proving possibilistic noninterference for a while language with parallel composition. We develop a uniform framework where we express a wide range of noninterference variants from the literature and compare them w.r.t. their *contracts*: the strength of the security properties they ensure weighed against the harshness of the syntactic conditions they enforce. This results in a simple implementable algorithm for proving that a program has a specific noninterference property, using only compositionality, which captures uniformly several security type-system results from the literature and suggests a further improved type system. All formalism and theorems have been mechanically verified in Isabelle/HOL.

1 Introduction

Language-based noninterference is an important and well-studied security property. To state this property, one 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. Then 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. Thus, 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.

Noninterference comes in several different variants, depending on what type of channels one accepts as capable of transmitting leaks—besides the normal channels represented by program variables, so-called *covert channels* include termination and timing channels. Moreover, when nondeterminism is involved, one can distinguish between *possibilistic* and *probabilistic* noninterference (the latter also taking probabilistic channels into account).

In this paper, we deal with noninterference in the presence of *possibilistic concurrency*. The literature abounds in notions of concurrent possibilistic noninterference and techniques to enforce it [2–6, 14, 19–21, 24, 29, 31], many of them surveyed in [23]. There is usually a tradeoff between the strength of a security property and the harshness of the conditions imposed on the programs in order to satisfy it (typically, a type system). Yet, new methods for establishing noninterference are often presented as improvements over older methods (e.g., a more lenient type system) while being rather brief on the notion that in effect the whole *contract* is being changed: less pressure on the programs, weaker noninterference ensured.

The paper presents the first comparison of a variety of noninterference notions and results, in a unified and formalized framework, where complex results from the literature are given uniform and simplified proofs. As a preview of the kind of properties we

* Supported by the DFG project Ni 491/13–1 (part of the DFG priority program RS3) and the DFG RTG 1480.

analyze and classify in this paper, here is a selection of informal notions of a command c being secure (noninterfering):

(1) Given any two initial memory states that are indistinguishable by the attacker (have the same low, i.e., public, part), the executions of c proceed identically w.r.t. both the program counter and the updates on the low part of the memory—we call this property *self isomorphism*.

(2) c may never change the low part of the memory during its execution—we call this *discreetness* (often in the literature this is called *highness*).

(3) If started in two indistinguishable memory states, the executions of c are lock-step bisimilar, performing the same updates to the low part of the memory—we call this *self strong bisimilarity*, i.e., strong bisimilarity to itself (called *strong security* in [25]).

(4) A relaxation of strong bisimilarity with lock-step synchronization replaced by *01-bisimilarity* (simply called *bisimilarity* in [5]), where only attacker-visible (i.e., low-memory changing) steps in one execution are required to be matched by corresponding steps in the other, while "discreet" (i.e., low-memory unchanging) steps need not be matched. Thus, one step may be matched by either zero or one steps.

(5) A further relaxation of strong bisimilarity—*weak bisimilarity* [16] (used in [4, 29] in a security context) where one step may be matched by any number of steps.

Property 1 (self isomorphism) is a very strong security notion, ensuring that an attacker controlling the low inputs of c is not able to infer any information about the high inputs, not even if he is allowed to observe the low part of intermediate memory states *and the program counter*. In particular, self isomorphism exhibits no leaks on covert channels such as timing or termination. Property 2 (discreetness) is neither weaker nor stronger than self isomorphism, but it no longer guarantees indistinguishability w.r.t. the program counter, and moreover the attacker may infer confidential information by measuring execution time. Property 3 (strong bisimilarity) prevents leaks on standard channels (low variable values) and timing channels, but, unlike self isomorphism, does not guarantee that execution starting in indistinguishable states follow the very same paths (taking the same branches). Properties 4 (01-bisimilarity) and 5 (weak bisimilarity) are weakenings of all of the above three. They are only able to guarantee the absence of leakage through standard channels.

Example 1 Consider the following commands, where l is a low variable and h, h' are high variables:

- c_0: $h := 0$
- c_1: $\mathsf{if}\ l = 0\ \mathsf{then}\ h := 1\ \mathsf{else}\ l := 2$
- c_2: $\mathsf{if}\ h = 0\ \mathsf{then}\ h := 1\ \mathsf{else}\ h := 2$
- c_3: $\mathsf{if}\ h = 0\ \mathsf{then}\ h := 1\ ;\ h := 2\ \mathsf{else}\ h := 3$
- c_4: $l := 4\ ;\ c_3$
- c_5: $c_3\ ;\ l := 4$
- c_6: $l := h$
- c_7: $h' := 0\ ;\ \mathsf{while}\ h > 0\ \mathsf{do}\ \{h := h - 1\ ;\ h' := h' + 1\}\ ;\ l := 4$

c_0 is both self isomorphic and discreet. c_1 is self isomorphic (since it is not testing any high variable), but not discreet. c_2 and c_3 are discreet (as they are not updating any low variable), but not self isomorphic. c_1 and c_2, but not c_3, are self strongly bisimilar—the reason why c_3 is not is its branching on a high test in conjunction with one branch

taking longer than the other. c_4 is self 01-bisimilar, because, after a self isomorphic assignment, it transits to a discreet continuation. c_5 is not self 01-bisimilar, but it is self weakly bisimilar. c_6 is not secure according to any of the five criteria— it exhibits a direct leak from high to low.

If we ignore timing channels and assume that initially $h \geq 0$, then it is reasonable to consider c_7 secure, since it has the same effect as the program $h' := h\,;\, l := 4$. However, whether or not we should deem c_7 secure *when placed in parallel with other threads* depends on the assumption we make on these threads—e.g., are they allowed to change h, thus preventing termination of c_7?

Note that the above example programs are sequential, which seems to contrast with our declared focus on concurrency—the explanation, hinted in the previous paragraph and detailed throughout the paper, is that the discussed notions of noninterference are defined anticipating parallel composition, i.e., so that the subject threads behave well when placed in parallel with other threads.

Here is an overview of this paper, where we use "security" and "noninterference" as synonyms. We start by introducing the concurrent setting where we operate: a while language with parallel composition and a fixed attacker-indistinguishability relation on program states (§2). Then we systematize and compare bisimilarity-based notions from the literature (§3). A formal study of the compositionality of, and of the implications between, these notions (§4) yields a novel proof methodology: To show that c is secure according to some notion N, first try to reduce the goal to proving N for the components of c; if this is not feasible due to failure of the required compositionality of N w.r.t. the language construct *Cns* located at the top of c (e.g., *Cns* can be an If, or a While, etc.), try to identify a stronger notion M that is (more) compositional w.r.t. *Cns*, and so on, recursively. The compositionality caveats of existing notions suggests the definition of a fully compositional security notion (§5). We then look at existing work on security type systems in the light of our analysis (§6)— the aforementioned simple proof technique turns out quite insightful, capturing these type system criteria uniformly. Our novel security notion from §5 yields a more permissive syntactic criterion than the existing ones, but the result targets only terminating programs. Finally, we discuss end-to-end security aspects of the studied bisimilarity-based notions (§7). We do not present any proofs of the stated facts—however, a (readable) Isabelle formalization of this paper's development, together with a map connecting the formal scripts with the propositions stated in this paper, is available at [18].

2 The programming language

We consider a simple while language with parallel composition, whose set **com** of commands, ranged over by c, d, e, is given by the following grammar:

$$c \;::=\; \mathit{atm} \mid \mathsf{Seq}\ c_1\ c_2 \mid \mathsf{If}\ \mathit{tst}\ c_1\ c_2 \mid \mathsf{While}\ \mathit{tst}\ c \mid \mathsf{Par}\ c_1\ c_2$$

Above, *atm* ranges over an unspecified set **atom** of atomic commands (atoms). Standard examples of atoms are assignments such as $x := x + y$. $\mathsf{Seq}\ c_1\ c_2$ is the sequential composition of c_1 and c_2, written in concrete syntax as $c_1\ ;\ c_2$. $\mathsf{If}\ \mathit{tst}\ c_1\ c_2$ is the conditional, written in concrete syntax as $\mathsf{if}\ \mathit{tst}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$, where *tst* ranges over an unspecified set **test** of tests. Standard examples of tests are Boolean expressions such as

$x = y$. While *tst c* is the usual while loop, in concrete syntax, while *tst* do c. Par c_1 c_2 is the parallel composition of c_1 and c_2, in concrete syntax, $c_1 \parallel c_2$. We generally prefer abstract syntax in theoretical results and concrete syntax in examples.

To give semantics to the language, we assume: a set of (memory) states, **state**, ranged over by s,t; an execution function for the atoms, aexec : **atom** $\to$ **state** $\to$ **state**; an evaluation function for the tests, tval : **test** $\to$ **state** $\to$ **bool**. Then we define a standard small-step semantics [17] as a pair of inductive predicates $\to_{\mathsf{T}}$: (**com** $\times$ **state**) $\to$ **state** and $\to_{\mathsf{C}}$: (**com** $\times$ **state**) $\to$ (**com** $\times$ **state**) (where the subscripts T and C stand for "termination" and "continuation") specified in Fig. 1. Intuitively, we interpret $(c,s)\to_{\mathsf{T}} s'$ as stating: in state s, command c may take a step terminating while changing the state to s'; and $(c,s)\to_{\mathsf{C}}(c',s')$ as saying: in state s, command c may take a step yielding the continuation c' while changing the state to s'. The pairs (c,s), which we call *configurations*, are thus thought of as consisting of the part of the program that remains to be executed, c, and the current state, s. We carefully distinguish between continuation and terminating steps (as the two predicates $\to_{\mathsf{C}}$ and $\to_{\mathsf{T}}$), since termination-sensitiveness will be crucial in our development. $\to_{\mathsf{C}}^*$ denotes the reflexive-transitive closure of $\to_{\mathsf{C}}$, and $\to_{\mathsf{T}}^*$ the composition of $\to_{\mathsf{C}}^*$ with $\to_{\mathsf{T}}$. Thus, $(c,s)\to_{\mathsf{C}}^*(c',s')$ means that (c',s') is reachable from (c,s) by zero or more continuation steps, and $(c,s)\to_{\mathsf{T}}^* s'$ that (the final state) s' is reachable from (c,s) by zero or more continuation steps followed by a terminating step.

$$(atm,s)\to_{\mathsf{T}} \mathsf{aexec}\ atm\ s \qquad \frac{(c_1,s)\to_{\mathsf{T}} s'}{(\mathsf{Seq}\ c_1\ c_2,s)\to_{\mathsf{C}}(c_2,s')} \qquad \frac{(c_1,s)\to_{\mathsf{C}}(c_1',s')}{(\mathsf{Seq}\ c_1\ c_2,s)\to_{\mathsf{C}}(\mathsf{Seq}\ c_1'\ c_2,s')}$$

$$\frac{\mathsf{tval}\ tst\ s}{(\mathsf{If}\ tst\ c_1\ c_2,s)\to_{\mathsf{C}}(c_1,s)} \qquad \frac{\neg\ \mathsf{tval}\ tst\ s}{(\mathsf{If}\ tst\ c_1\ c_2,s)\to_{\mathsf{C}}(c_2,s)} \qquad \frac{\neg\ \mathsf{tval}\ tst\ s}{(\mathsf{While}\ tst\ c,s)\to_{\mathsf{T}} s}$$

$$\frac{(c_1,s)\to_{\mathsf{C}}(c_1',s')}{(\mathsf{Par}\ c_1\ c_2,s)\to_{\mathsf{C}}(\mathsf{Par}\ c_1'\ c_2,s')} \qquad \frac{\mathsf{tval}\ tst\ s}{(\mathsf{While}\ tst\ c,s)\to_{\mathsf{C}}(\mathsf{Seq}\ c\,(\mathsf{While}\ tst\ c),s)}$$

$$\frac{(c_2,s)\to_{\mathsf{C}}(c_2',s')}{(\mathsf{Par}\ c_1\ c_2,s)\to_{\mathsf{C}}(\mathsf{Par}\ c_1\ c_2',s')} \qquad \frac{(c_2,s)\to_{\mathsf{T}} s'}{(\mathsf{Par}\ c_1\ c_2,s)\to_{\mathsf{C}}(c_1,s')} \qquad \frac{(c_1,s)\to_{\mathsf{T}} s'}{(\mathsf{Par}\ c_1\ c_2,s)\to_{\mathsf{C}}(c_2,s')}$$

Fig. 1: Small-step semantics

3 Notions of noninterference

Next we proceed to a uniform description of several notions of noninterference from the literature. 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 Often, $\sim$ is defined as follows. We assume that atomic statements and tests are built by means of arithmetic and boolean expressions applied to variables taken from a set **var**. States are assignments of values to variables, i.e., the set **state** is **var** $\to$ **val**, where **val** is a set of values (e.g., integers). Variables are classified as either low (lo) or high (hi) by a given security level function sec : **var** $\to \{\mathsf{lo},\mathsf{hi}\}$. Then $\sim$ is defined as coincidence on the low variables, with the intuition that the attacker is only able to observe these. Formally, $s \sim t \equiv \forall x \in \mathbf{var}.\ \mathsf{sec}\ x = \mathsf{lo} \Longrightarrow s\,x = t\,x$.

We define the following predicates on commands *coinductively as greatest fixed points*, i.e., as the *strongest* predicates satisfying the indicated clauses:

– *Self isomorphism*, siso, by $\mathsf{siso}\ c \equiv$
$(\forall s\ t\ c'\ s'.\ s \sim t\ \wedge\ (c,s) \rightarrow_{\mathsf{c}} (c',s') \Longrightarrow (\exists t'.\ (c,t) \rightarrow_{\mathsf{c}} (c',t')\ \wedge\ s' \sim t'))\ \wedge$
$(\forall s\ t\ s'.\ s \sim t\ \wedge\ (c,s) \rightarrow_{\mathsf{T}} s' \Longrightarrow (\exists t'.\ (c,t) \rightarrow_{\mathsf{T}} t'\ \wedge\ s' \sim t'))\ \wedge$
$(\forall s\ c'\ s'.\ (c,s) \rightarrow_{\mathsf{c}} (c',s') \Longrightarrow \mathsf{siso}\ c')$
– *Discreetness*, discr, by $\mathsf{discr}\ c \equiv$
$(\forall s\ c'\ s'.\ (c,s) \rightarrow_{\mathsf{c}} (c',s') \Longrightarrow s \sim s'\ \wedge\ \mathsf{discr}\ c') \wedge (\forall s\ s'.\ (c,s) \rightarrow_{\mathsf{T}} s' \Longrightarrow s \sim s')$

The coinductive definition of self isomorphism expresses that that execution of a command proceeds absolutely independently of the indistinguishability class of the state, and this is true *interactively*, i.e., *regardless of the intervention of the environment*, provided this intervention is itself compatible with the state indistinguishability relation. And similarly for the definition of discr, expressing that the command never changes the indistinguishability class, regardless of what that class has become due to potential action from the environment.

The last aspect, interactivity, is expressed by the universal quantification over the indistinguishable states s and t in the definition of siso. Indeed, even though transitions operate on (command, state) pairs, the siso predicate operates on commands alone, forgetting each time the result state s' from the continuation (c',s'). Thus, at each resumption point, the predicate quantifies universally over *all* states s ("overwriting" the previous s'), to account for the fact that the new state produced by the command under consideration may have been changed by the environment (perhaps consisting of other threads running in parallel, and/or of the attacker) before that command gets to perform an other step. For example, the command $c \equiv h := 0\ ;\ l := h$ (with h high and l low) would be deemed as self isomorphic if it were not for the interactivity constraint. Indeed, if no interference from the environment is assumed, the execution of c proceeds the same way regardless of the initial value of h, as it first assigns 0 to h. However, $\mathsf{siso}\ c$ does not hold, since the continuation $l := h$ is required to be secure *given any value of h* arising as the effect of a secure thread running in parallel, say, $h := h'$ with h' high. This interactivity twist (originating from [22, 25]) is convenient for compositionality, since it ensures that a command is secure not only in isolation, but also if placed in any pool of secure threads running in parallel. As a consequence, most of the security notions discussed in this paper will be interactive.

We shall also need the following interactive notion of termination possibility at each point during execution, via the coinductively defined predicate mayT (read "may terminate"): $\mathsf{mayT}\ c \equiv \forall s\ c'\ s'.(c,s) \rightarrow_{\mathsf{c}} (c',s') \Longrightarrow (\exists s''.(c',s') \rightarrow_{\mathsf{T}}^{*} s'') \wedge \mathsf{mayT}\ c'$.

Self isomorphism and discreetness were expressible as unary predicates. However, interesting noninterference properties may require binary relations. To see this, assume we wish to express that c is secure, i.e., its executions are (multi)step-wise equivalent if started in indistinguishable states. Assume c branches according to a high test. Then indistinguishable states may yield different continuations, say, c_1 and c_2, and so we are faced with the problem of proving the executions of c_1 and c_2 (multi)step-wise equivalent, i.e., proving c_1 and c_2 *bisimilar*. (The above two notions have by-passed this problem in trivial ways: self isomorphism forbids this situation by disallowing the program counter to diverge, hence disallowing high tests, while discreetness of c also requires c_1 and c_2 to be discreet, hence trivially "equivalent".)

In order to define relevant notions of bisimilarity, it will be useful to first introduce matching operators (or *matchers*) that express various choices of rules for the bisimilar-

$\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}\ \theta\ c\ d \equiv$
$\forall s\ t\ c'\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{C}} (c',s') \Longrightarrow$
$(\exists d'\ t'.\ (d,t) \rightarrow_{\mathsf{C}} (d',t') \wedge s' \sim t' \wedge \theta\ c'\ d')$

$\mathsf{match}^{\mathsf{C}}_{\mathsf{01C}}\ \theta\ c\ d \equiv$
$\forall s\ t\ c'\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{C}} (c',s') \Longrightarrow$
$(\exists d'\ t'.\ (d,t) \rightarrow_{\mathsf{C}} (d',t') \wedge s' \sim t' \wedge \theta\ c'\ d') \vee$
$(s' \sim t \wedge \theta\ c'\ d)$

$\mathsf{match}^{\mathsf{C}}_{\mathsf{01}}\ \theta\ c\ d \equiv$
$\forall s\ t\ c'\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{C}} (c',s') \Longrightarrow$
$(\exists d'\ t'.\ (d,t) \rightarrow_{\mathsf{C}} (d',t') \wedge s' \sim t' \wedge \theta\ c'\ d') \vee$
$(s' \sim t \wedge \theta\ c'\ d) \vee$
$(\exists t'.\ (d,t) \rightarrow_{\mathsf{T}} t' \wedge s' \sim t' \wedge \mathsf{discr}\ c')$

$\mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d \equiv$
$\forall s\ t\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{T}} s' \Longrightarrow$
$(\exists t'.\ (d,t) \rightarrow_{\mathsf{T}} t' \wedge s' \sim t')$

$\mathsf{match}^{\mathsf{T}}_{\mathsf{01}}\ c\ d \equiv$
$\forall s\ t\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{T}} s' \Longrightarrow$
$(\exists t'.\ (d,t) \rightarrow_{\mathsf{T}} t' \wedge s' \sim t') \vee$
$(\exists d'\ t'.\ (d,t) \rightarrow_{\mathsf{C}} (d',t') \wedge s' \sim t' \wedge \mathsf{discr}\ d') \vee$
$(s' \sim t \wedge \mathsf{discr}\ d)$

$\mathsf{match}^{\mathsf{C}}_{\mathsf{MC}}\ \theta\ c\ d \equiv$
$\forall s\ t\ c'\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{C}} (c',s') \Longrightarrow$
$(\exists d'\ t'.\ (d,t) \rightarrow^*_{\mathsf{C}} (d',t') \wedge s' \sim t' \wedge \theta\ c'\ d')$

$\mathsf{match}^{\mathsf{C}}_{\mathsf{M}}\ \theta\ c\ d \equiv$
$\forall s\ t\ c'\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{C}} (c',s') \Longrightarrow$
$(\exists d'\ t'.\ (d,t) \rightarrow^*_{\mathsf{C}} (d',t') \wedge s' \sim t' \wedge \theta\ c'\ d') \vee$
$(\exists t'.\ (d,t) \rightarrow^*_{\mathsf{T}} t' \wedge s' \sim t' \wedge \mathsf{discr}\ c')$

$\mathsf{match}^{\mathsf{T}}_{\mathsf{MT}}\ c\ d \equiv$
$\forall s\ t\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{T}} s' \Longrightarrow$
$(\exists t'.\ (d,t) \rightarrow^*_{\mathsf{T}} t' \wedge s' \sim t')$

$\mathsf{match}^{\mathsf{T}}_{\mathsf{M}}\ c\ d \equiv$
$\forall s\ t\ s'.\ s \sim t \wedge (c,s) \rightarrow_{\mathsf{T}} s' \Longrightarrow$
$(\exists t'.\ (d,t) \rightarrow^*_{\mathsf{T}} t' \wedge s' \sim t') \vee$
$(\exists d'\ t'.\ (d,t) \rightarrow^*_{\mathsf{C}} (d',t') \wedge s' \sim t' \wedge \mathsf{discr}\ d')$

Fig. 2: Matchers

ity game. They are defined in Fig. 2, where θ ranges over binary relations on commands. In the operator names, the superscripts indicate the kind of steps being taken, and the subscripts indicate by what kind of steps these must be simulated (matched), where: "C" means (single) continuation step; "T" means (single) terminating step; "01C" means 0 or 1 continuation steps; "01" means 0 or 1 continuation or terminating steps, i.e., 01C or T; "MC" means multiple continuation steps; "MT" means multiple continuation steps, followed by a terminating step; "M" means MC or MT. E.g., $\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}$ refers to matching any continuation step by a continuation step, $\mathsf{match}^{\mathsf{C}}_{\mathsf{01C}}$ to matching any continuation step by 0 or 1 continuation steps, i.e., either by a continuation step or by a stutter move.

Matchers indicate how the single steps of a command c may be matched by *single or multiple* steps of a command d. In most cases, the matcher is also parameterized by a continuation relation θ; exceptions are $\mathsf{match}^{\mathsf{T}}_{\mathsf{T}}$ and $\mathsf{match}^{\mathsf{T}}_{\mathsf{MT}}$, where, due to termination of both the left and the right sides, no continuation makes sense. $\mathsf{match}^{\mathsf{C}}_{\mathsf{01}}$, $\mathsf{match}^{\mathsf{T}}_{\mathsf{01}}$, $\mathsf{match}^{\mathsf{C}}_{\mathsf{M}}$ and $\mathsf{match}^{\mathsf{T}}_{\mathsf{M}}$ are termination-flexible matchers, in that they allow matching continuation steps against termination steps and vice versa. For instance, $\mathsf{match}^{\mathsf{C}}_{\mathsf{01}}$ ("match a continuation step against 0 or 1 steps of either kind") requires for θ, c and d that, for all indistinguishable states s and t, any step $(c,s) \rightarrow_{\mathsf{C}} (c',s')$ be matched by either a continuation step $(d,t) \rightarrow_{\mathsf{C}} (d',t')$, or a stutter step, or a termination step $(d,t) \rightarrow_{\mathsf{T}} t'$. In each case, it is also required that the resulting states are indistinguishable. Moreover, in the first two cases (for continuation and stutter) it is required that the resulting commands are in relation θ. For the third case though (the termination step), the latter condition does not make sense, since on the left of the matcher we have a continuation command, c', while the right side has terminated; what we require instead is that, w.r.t. the attacker-observable behavior, c' acts as if it terminated, in that it will never change the

indistinguishability class of the state, i.e., is discreet. (Similar discreetness conditions appear in the definitions of the other termination-flexible matchers for similar reasons.)

We are now ready to define the following bisimilarity relations, again coinductively, by plugging in different combinations of matchers and taking each time the *largest symmetric relation* satisfying the given clause (where the bisimilarities are written with infix notation on the left and are passed as arguments to the matchers on the right):

- *Strong bisimilarity*, $\approx_{\mathsf{S}}$, by $c \approx_{\mathsf{S}} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{C}}\ (\approx_{\mathsf{S}})\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d$
- *01-bisimilarity*, $\approx_{01}$, by $c \approx_{01} d \equiv \mathsf{match}^{\mathsf{C}}_{01}\ (\approx_{01})\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{01}\ c\ d$
- *Termination-sensitive 01-bisimilarity* (*01T-bisimilarity*), $\approx_{01\mathsf{T}}$, by $c \approx_{01\mathsf{T}} d \equiv \mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ (\approx_{01\mathsf{T}})\ c\ d \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d$
- *Weak bisimilarity*, $\approx_{\mathsf{W}}$, by $c \approx_{\mathsf{W}} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{M}}\ (\approx_{\mathsf{W}})\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{M}}\ c\ d$
- *Termination-sensitive weak bisimilarity* (*weak T-bisimilarity*), $\approx_{\mathsf{WT}}$, by $c \approx_{\mathsf{W}} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{MC}}\ (\approx_{\mathsf{W}})\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{MT}}\ c\ d$

All these bisimilarity relations are by definition symmetric and can also be proved transitive, but they are *not* reflexive. In fact, the notion of a command c being bisimilar with itself (e.g., $c \approx_{\mathsf{S}} c$, $c \approx_{01} c$, etc.), which we call *self bisimilarity of c* (e.g., self strong bisimilarity, self 01-bisimilarity, etc.) is taken in this paper as the formalization of the informal notion of security of a command. Below we explain how different bisimilarities correspond to different attacker models.

In all cases, one assumes the attacker has access to the program (command) source code and the low part of the state, and the ability to set, at the beginning of the command execution, the low part of the state in any desired way. For strong bisimilarity ($\approx_{\mathsf{S}}$), we assume the attacker's ability to repeatedly stop the program after single execution steps and inspect the (low part of the) state, or, equivalently, take snapshots of the state after controlled numbers of execution steps. Technically, this shows in the two involved matchers, $\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}$ and $\mathsf{match}^{\mathsf{T}}_{\mathsf{T}}$, being one-to-one (w.r.t. continuation or termination steps). Moreover, we assume the attacker can detect termination—this shows in the fact that the two matchers preserve the type of transition: continuation vs. continuation and termination vs. termination. For weak bisimilarity ($\approx_{\mathsf{W}}$), the attacker may still stop the program repeatedly, but has no control on the number of steps that the program takes between two stops. (For what the attacker knows, zero, one, or more steps could have been taken.) This shows in the one-to-many nature of the matchers. The termination-sensitive version of weak bisimilarity ($\approx_{\mathsf{WT}}$) additionally assumes the attacker is able to detect termination. Thus, $\approx_{\mathsf{WT}}$ allows, via $\mathsf{match}^{\mathsf{T}}_{\mathsf{MT}}$, matching a termination step by a sequence of steps only if the latter ends in a termination step. 01-bisimilarity ($\approx_{01}$), also coming with a termination-sensitive variant ($\approx_{01\mathsf{T}}$), is intermediate between strong and weak bisimilarity. Here, the attacker may keep running the program for 0 or 1 steps, without knowing which of the two situations has actually occurred.

The following proposition, relating different notions of self bisimilarity, follows easily from the definitions of the corresponding matchers:

Prop 1 The implications in Fig. 3 hold.

Note that discreetness implies self 01-bisimilarity, but not self 01T-bisimilarity. However, for may-terminating processes (roughly, processes with finite behavior), it does imply self wT-bisimilarity.

Example 1 already illustrates most of the above bisimilarities. Here are some further illustrations that also take Prop. 1 into account (using the Example 1 notations).

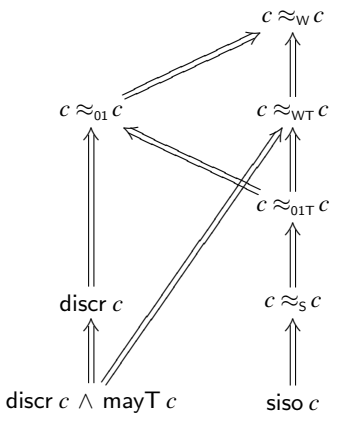

Fig. 3: Implications between security notions

c	mayT c	discr c	$\varphi\ c$	$\psi\ c$
atm	True	pres *atm*	cpt *atm*	cpt *atm*
Seq $c_1\ c_2$	mayT c_1 mayT c_2	discr c_1 discr c_2	$\varphi\ c_1$ $\varphi\ c_2$	$\psi_{\mathsf{T}}\ c_1$ $\psi\ c_2$ $\psi\ c_1$ discr c_2
If *tst* $c_1\ c_2$	mayT c_1 mayT c_2	discr c_1 discr c_2	cpt *tst* $\varphi\ c_1$ $\varphi\ c_2$	cpt *tst* $\psi\ c_1$ $\psi\ c_2$
While *tst* d	False	discr d	cpt *tst* $\varphi\ d$	False
Par $c_1\ c_2$	mayT c_1 mayT c_2	discr c_1 discr c_2	$\varphi\ c_1$ $\varphi\ c_2$	$\psi\ c_1$ $\psi\ c_2$

$\varphi \in \{\mathsf{siso}, \approx_{\mathsf{S}}, \approx_{01\mathsf{T}}, \approx_{\mathsf{WT}}\} \quad \psi \in \{\approx_{01}, \approx_{\mathsf{W}}\}$

$$\psi_{\mathsf{T}} \equiv \begin{cases} \approx_{01\mathsf{T}}, \text{ if } \psi = \approx_{01} \\ \approx_{\mathsf{WT}}, \text{ if } \psi = \approx_{\mathsf{W}} \end{cases}$$

Fig. 4: Compositionality table

Example 3 (1) c_3 is self 01-bisimilar, as any two discreet processes are 01-bisimilar.

(2) However, c_3 is not self 01T-bisimilar, as shown by the following reasoning: depending on h, c_3 can move to $d \equiv h := 1$; $h := 2$ or $e \equiv h := 3$; but d and e are not 01T-bisimilar, as d is not able to 01T-match the immediate terminating step from e.

(3) The above is not a problem for weak T-bisimilarity though, since here d can catch up with e by taking multiple steps. Thus, c_3 is self weakly T-bisimilar (as any two discreet processes with finite behavior are weakly T-bisimilar).

(4) $c_5 \equiv c_3$; $l := 4$ is self weakly T-bisimilar, since alternative executions (starting in indistinguishable states) of its first part c_3 are able to $\approx_{\mathsf{WT}}$-synchronize, so that they can proceed strongly synchronously with the remaining non-discreet step $l := 4$.

(5) However, c_5 is not self 01-bisimilar since the above e-continuation of c_3 is able to terminate first, putting itself in a position to take the non-discreet step $l := 4$, not available at that time for the other continuation, d.

(6) while $h = 0$ do $h := 0$ is discreet, hence self 01-bisimilar, but not weakly T-bisimilar, as a diverging execution from $h = 0$ cannot match a terminating one from $h \neq 0$.

The weak and 01-bisimilarities provide the most fruitful notions in type-system approaches to noninterference. (The others—self isomorphism, discreetness and strong bisimilarity—are too harsh requirements, but, as we shall see, turn out as useful auxiliaries.) Thus, Smith and Volpano [29] focus on termination-sensitive weak bisimilarity. On the other hand, Boudol and Castellani [5, 6] prefer termination-insensitive 01-bisimilarity, while later Boudol [4] also considers weak bisimilarity, but in its termination-insensitive form. In these works, the newly introduced bisimilarities are not formally compared with preexisting ones—instead, the focus is on comparing the end-product type systems, i.e., the *rely* side of the contract (while the bismilarities are the *guarantee* part). In order to properly revisit and compare type-system results, we first need an analysis of compositionality for these bisimilarities.

4 Compositionality

We now move to the central concept of this paper—compositionality of noninterference w.r.t. the language constructs.

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 $\forall s\ t.\ s \sim t \Longrightarrow \mathsf{tval}\ tst\ s = \mathsf{tval}\ tst\ t$. 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 (high expressions). Moreover, for tests, $\sim$-compatibility means no dependence on high variables.

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

Here is how to read Fig. 4. 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 3 column 3 says: if $\mathsf{discr}\ c_1$ and $\mathsf{discr}\ c_2$ then $\mathsf{discr}\ (\mathsf{Seq}\ c_1\ c_2)$. The horizontal line in row 3 column 5 represents an "or" – thus, row 3 column 5 says: if either $[\psi_{\mathsf{T}}\ c_1$ and $\psi\ c_2]$ or $[\psi\ c_1$ and $\mathsf{discr}\ c_2]$ then $\psi\ (\mathsf{Seq}\ c_1\ c_2)$. The involved bisimilarities are considered in their unary, "self" form, e.g., $\psi\ c$ means $c \approx_{01} c$ or $c \approx_{\mathsf{W}} c$.

Example 4 The informal arguments in Examples 1 and 3 can be made rigorous using the compositionality table in Fig. 4 in conjunction with the implication graph in Fig. 3. For instance, c_4 from Example 1 has the form $\mathsf{Seq}\ (l := 4)\ c_3$, where c_3 has the form $\mathsf{If}\ (h = 0)\ (\mathsf{Seq}\ (h := 1)\ (h := 2))\ (h := 3)$. According to the table, for $c_4 \approx_{01} c_4$, it suffices that $(l := 4) \approx_{01\mathsf{T}} (l := 4)$ and $c_3 \approx_{01} c_3$. The former is true by the table, since $l := 4$ is compatible. However, the table cannot help (yet) in proving $c_3 \approx_{01} c_3$, because there the required side condition is $\mathsf{cpt}\ (h = 0)$, which does not hold. Therefore we turn to the implication graph, and try to prove the fact for one of the predecessors of $\approx_{01}$. One predecessor is $\approx_{01\mathsf{T}}$, which again requires $\mathsf{cpt}\ (h = 0)$, and so does its predecessor $\approx_{\mathsf{S}}$, and so does the predecessor of the latter, siso, which is a bottom node—therefore this path fails. The other predecessor of $\approx_{01}$ is discr, for which the table does not require the problematic side-condition. And the proof of $\mathsf{discr}\ c_3$ goes smoothly according to the table, since it is reduced to $\mathsf{discr}\ (\mathsf{Seq}\ (h := 1)\ (h := 2))$ and $\mathsf{discr}\ (h := 3)$, and further to $\mathsf{discr}\ (h := 1)$, $\mathsf{discr}\ (h := 2)$ and $\mathsf{discr}\ (h := 3)$, all being true by $\sim$-preservation.

Note that we appeal to the Fig. 3 graph whenever the table result is not sufficiently strong, i.e., the given security notion is not sufficiently compositional w.r.t. the given language construct. For this table-and-graph proof technique, it is instructive to compare the termination-sensitive security notions with the termination-insensitive ones, that is, φ with ψ in Fig. 4. φ is more compositional than ψ w.r.t. Seq. (In fact, if interactivity is responsible for Par-compositionality, termination-sensitiveness can be deemed responsible for Seq-compositionality.) Indeed, for $\psi\ (\mathsf{Seq}\ c_1\ c_2)$ to go through, the table requires strengthening ψ either for c_1 to its termination-sensitive variant, ψ_{T}, or for c_2 to discreetness. A consequence of this is also the lack of compositionality of ψ w.r.t. While (since the semantics of While involves iteration of Seq). On the other hand, ψ enjoys better compositionality w.r.t. If. This is not visible by looking at the table alone, where the If rules of φ and ψ are the same, and they are both conditioned by

the $\sim$-compatibility of *tst*. The difference appears when *tst* is not compatible—then, according to the graph, unlike φ, ψ can "fall back" on discr, which does not require *tst* to be compatible. Indeed, unlike φ, ψ is above discr in the graph. Note that, among the φ's, $\approx_{\mathsf{WT}}$ is the best located with this respect, since it is above the conjunction of $\mathsf{discr}\ c$ and $\mathsf{mayT}\ c$ in the graph. But this is still worse than ψ, since falling back on $\mathsf{discr}\ c \wedge \mathsf{mayT}\ c$ forbids while loops, as shown in the table for mayT.

An interesting theoretical question is whether we can have the best of both worlds and define a relation that is both above discreetness in the graph and fully compositional w.r.t. Seq, without sacrificing compositionality with the other constructs. A positive answer to this question is presented next.

5 A more compositional security notion

The rough idea of the proposed solution is as follows. If we knew that the whole program terminates, then discreetness would imply $\approx_{\mathsf{WT}}$. And to integrate termination information into our coinductive interactiveness, we note that, given a thread c running in parallel with others in a pool whose execution from a given state s is known to terminate, the following are true: (1) the execution of c alone starting in s must terminate; (2) between resumption points of the execution of c, the other threads are guaranteed to change the state in such a way that termination is preserved. This leads us to $\approx_{\mathsf{T}}$, a relaxation of $\approx_{\mathsf{WT}}$ with interactivity restricted to mustT ("must terminate") configurations, where $\mathsf{mustT}(c,s)$ is defined to mean that there exists no infinite chain $(c_0,s_0),\ldots,(c_n,s_n),\ldots$ such that $(c_0,s_0)=(c,s)$ and $\forall i.\ (c_i,s_i)\rightarrow_{\mathsf{c}}(c_{i+1},s_{i+1})$:

- $\mathsf{match}^{\mathsf{C}}_{\mathsf{TMC}}\ \theta\ c\ d \equiv \forall s\ t\ c'\ s'.\ \mathsf{mustT}\ (c,s) \wedge \mathsf{mustT}\ (d,t) \wedge s \sim t \wedge (c,s)\rightarrow_{\mathsf{c}}(c',s') \Longrightarrow (\exists d'\ t'.\ (d,t)\rightarrow^*_{\mathsf{c}}(d',t') \wedge s' \sim t' \wedge \theta\ c'\ d')$
- $\mathsf{match}^{\mathsf{T}}_{\mathsf{TMT}}\ c\ d \equiv \forall s\ t\ s'.\ \mathsf{mustT}\ (c,s) \wedge \mathsf{mustT}\ (d,t) \wedge s \sim t \wedge (c,s)\rightarrow_{\mathsf{T}}s' \Longrightarrow (\exists t'.\ (d,t)\rightarrow^*_{\mathsf{T}}t' \wedge s' \sim t')$
- $c \approx_{\mathsf{T}} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{TMC}}\ (\approx_{\mathsf{T}})\ c\ d \wedge \mathsf{match}^{\mathsf{T}}_{\mathsf{TMT}}\ c\ d$

And, indeed, $\approx_{\mathsf{T}}$ achieves the targeted properties, as can be shown by an argument similar to those of Props. 1 and 2:

Prop 3 (1) The compositionality facts stated in Fig. 4 for φ also hold for $\approx_{\mathsf{T}}$.
(2) $\mathsf{discr}\ c \Longrightarrow c \approx_{\mathsf{T}} c$.

Note that $\approx_{\mathsf{T}}$ does not require, for the involved programs, termination (a liveness property), but rather preservation of termination (a safety property). $\approx_{\mathsf{T}}$ is weaker than $\approx_{\mathsf{WT}}$, and neither weaker nor stronger than $\approx_{01}$ and $\approx_{\mathsf{W}}$. The benefit of having $\approx_{\mathsf{T}}$ better suited than the other notions w.r.t. our table-and-graph reasoning is the availability of a more permissive syntactic criterion, as we detail next.

6 Syntactic criteria

The (compositionality based) table-and-graph proof technique described in Example 4 can be automated, yielding a collection of recursive syntactic predicates corresponding to the various security notions. The recursive clauses for these predicates will simply perform the necessary lookups: first in the table, then, if needed, in the graph.

Before listing these clauses, we first simplify the Fig. 3 graph, noticing that $\approx_{\mathsf{S}}$ and $\approx_{01\mathsf{T}}$ are redundant nodes on top of siso. Indeed, the compositionality conditions for $\approx_{\mathsf{S}}$

and $\approx_{01\mathsf{T}}$ from the Fig. 4 table are identical to those of all nodes below, hence identical to those of siso. This means that, when proving $c \approx_{\mathsf{S}} c$ or $c \approx_{01\mathsf{T}} c$, one cannot do better than proving compositionality of the stronger (more desirable) siso notion of security. We therefore drop $\approx_{\mathsf{S}}$ and $\approx_{01\mathsf{T}}$ from the graph. Fig. 5 shows this new graph, where $\approx_{\mathsf{T}}$ is also integrated. In the Fig. 4 table, we also redefine ψ_{T} by redirecting $\approx_{01}$ to siso:

$$\psi_{\mathsf{T}} \equiv \begin{cases} \mathsf{siso}, \text{ if } \psi = \approx_{01} \\ \approx_{\mathsf{WT}}, \text{ if } \psi = \approx_{\mathsf{W}}. \end{cases}$$

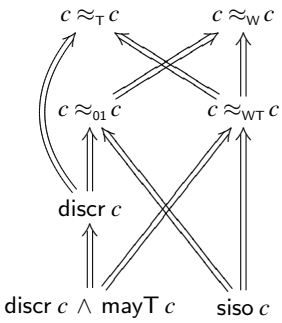

Fig. 5: Simplified implication graph of security notions

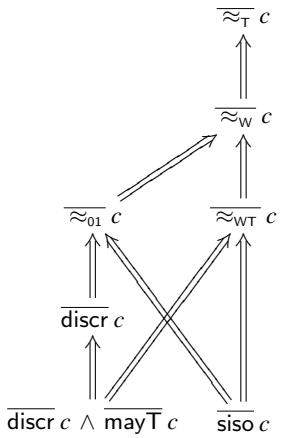

Fig. 6: Syntactic implications

Let us introduce some notation for the Fig. 4 table and the Fig. 5 graph. A (syntactic) *constructor Cns* is any of the following: Seq, If *tst* where *tst* $\in$ **test**, While *tst* where *tst* $\in$ **test**, Par. In addition, for uniformity, we also introduce a constructor Atm *atm* for every *atm* $\in$ **atom**, and assume Atm *atm* is the same as *atm*. Thus, any command c has the form *Cns* $c_1 \ldots c_k$, where *Cns* is a constructor and $c_1 \ldots c_k$ are k commands, the *components* of c, with k either 0, 1 or 2, depending on *Cns* (it is 0 for Atm *atm*).

Henceforth, we let χ range over the notions in the table, namely, $\chi \in \{\mathsf{mayT}, \mathsf{discr}, \mathsf{siso}, \approx_{\mathsf{S}}, \approx_{01\mathsf{T}}, \approx_{\mathsf{WT}}, \approx_{01}, \approx_{\mathsf{W}}, \approx_{\mathsf{T}}\}$. The table has an entry corresponding to every combination (χ, Cns), for which we define the following:

- $\mathsf{side}_{\chi,\mathit{Cns}}$ is its *side condition*, i.e., the part of it not depending on the components. If this part is empty, we put True. E.g., $\mathsf{side}_{\mathsf{mayT},\mathsf{Atm}\ \mathit{atm}} = \mathsf{side}_{\mathsf{siso},\mathsf{Seq}} = \mathsf{True}$, $\mathsf{side}_{\mathsf{siso},\mathsf{If}\ \mathit{tst}} = \mathsf{cpt}\ \mathit{tst}$.
- $\mathsf{rcond}_{\chi,\mathit{Cns}}(c_1, \ldots, c_k)$ is its *recursion condition*, i.e., the part involving the components of c. Again, if this part is empty, we put True. E.g., $\mathsf{rcond}_{\mathsf{mayT},\mathsf{Atm}\ \mathit{atm}} = \mathsf{True}$, $\mathsf{rcond}_{\mathsf{siso},\mathsf{Seq}}(c_1, c_2) = \mathsf{rcond}_{\mathsf{siso},\mathsf{If}\ \mathit{tst}}(c_1, c_2) = (\mathsf{siso}\ c_1 \wedge \mathsf{siso}\ c_2)$.

For any element χ in the graph, we let Pred χ denote its set of predecessors. E.g., Pred siso $= \emptyset$, Pred $\approx_{01} = \{\mathsf{discr}, \mathsf{siso}\}$, Pred $\approx_{\mathsf{W}} = \{\approx_{01}, \approx_{\mathsf{WT}}\}$.

Note that, for all χ, *Cns*, and c of the form *Cns* $c_1 \ldots c_k$,

- The table ensures that $\mathsf{side}_{\chi,\mathit{Cns}} \wedge \mathsf{rcond}_{\chi,\mathit{Cns}}(c_1, \ldots, c_k) \Longrightarrow \chi\ c$;
- The graph ensures that $(\bigvee_{\chi' \in \mathsf{Pred}\ \chi} \chi'\ c) \Longrightarrow \chi\ c$.

We define, for each security notions χ, a syntactic predicate $\overline{\chi}$ on commands by turning the above implications into recursive clauses for each constructor *Cns*, where one first tries the table, and then, if the table fails, one tries the graph:

$$\overline{\chi}\ (Cns\ c_1 \ldots c_k) \equiv \begin{cases} \overline{\mathsf{rcond}_{\chi,Cns}}(c_1,\ldots,c_k), & \text{if } \overline{\mathsf{side}_{\chi,Cns}}(c_1,\ldots,c_k), \\ \bigvee_{\chi' \in \mathsf{Pred}\,\chi} \overline{\chi'}\ (Cns\ c_1 \ldots c_k), & \text{otherwise,} \end{cases}$$

where $\overline{\mathsf{rcond}_{\chi,Cns}}$ and $\overline{\mathsf{side}_{\chi,Cns}}$ are $\mathsf{rcond}_{\chi,Cns}$ and $\mathsf{side}_{\chi,Cns}$ with all the involved security predicates χ' replaced by their syntactic counterparts $\overline{\chi'}$.

For example, taking $Cns = \mathsf{If}\ tst$, we have:

1. $\overline{\mathsf{discr}}\ (\mathsf{If}\ tst\ \ c_1\ c_2) = (\overline{\mathsf{discr}}\ c_1\ \wedge\ \overline{\mathsf{discr}}\ c_2)$.
2. $\overline{\mathsf{siso}}\ (\mathsf{If}\ tst\ c_1\ c_2) = \begin{cases} \overline{\mathsf{siso}}\ c_1\ \wedge\ \overline{\mathsf{siso}}\ c_2, & \text{if } \mathsf{cpt}\ tst \\ \mathsf{False}, & \text{otherwise.} \end{cases}$
3. $\overline{\approx_{01}}\ (\mathsf{If}\ tst\ c_1\ c_2) = \begin{cases} \overline{\approx_{01}}\ c_1\ \wedge\ \overline{\approx_{01}}\ c_2, & \text{if } \mathsf{cpt}\ tst \\ \overline{\mathsf{discr}}\ (\mathsf{If}\ tst\ c_1\ c_2)\ \vee\ \overline{\mathsf{siso}}\ (\mathsf{If}\ tst\ c_1\ c_2), & \text{otherwise} \end{cases}$

   $= (\text{by 1 and 2}) = \begin{cases} \overline{\approx_{01}}\ c_1\ \wedge\ \overline{\approx_{01}}\ c_2, & \text{if } \mathsf{cpt}\ tst \\ \overline{\mathsf{discr}}\ c_1\ \wedge\ \overline{\mathsf{discr}}\ c_2, & \text{otherwise.} \end{cases}$

(Recall that, when we instantiate χ to a bisimilarity such as $\approx_{\mathsf{WT}}$, we refer to its unary version, taking $\chi\ c$ to be $c \approx_{\mathsf{WT}} c$. Hence, an instance of $\overline{\chi}$ is the unary predicate $\overline{\approx_{\mathsf{WT}}}$.)

The proof of the following fact is now routine by structural induction on commands:

Prop 4 The syntactic criteria $\overline{\chi}$ are sound for the security notions χ in Fig. 5, in that $\overline{\chi}\ c \Longrightarrow \chi\ c$ for all commands c.

A remarkable property of the $\overline{\chi}$'s is that they preserve the Fig. 5 hierarchy of χ's. In fact, they actually refine it:

Prop 5 The implications listed in Fig. 6 hold.

The hierarchy refinement from Fig. 5 to Fig. 6 consists of the advance of $\overline{\approx_{\mathsf{T}}}$ to the top, even though $\approx_{\mathsf{T}}$ is not weaker than $\approx_{\mathsf{W}}$ or $\approx_{01}$. The reason why $\overline{\approx_{\mathsf{T}}}$ is weaker than $\overline{\approx_{\mathsf{W}}}$ is the following: The recursive definition of each $\overline{\chi}$ is as *permissive* as is χ *compositional*. And since $\approx_{\mathsf{T}}$ is at least as compositional as $\approx_{\mathsf{W}}$ and any other relation involved in its compositionality (here, $\approx_{\mathsf{WT}}$), the proof of $\overline{\approx_{\mathsf{W}}}\ c \Longrightarrow \overline{\approx_{\mathsf{T}}}\ c$ goes through by structural induction on c.

So far, our analysis was purely semantic and local: for semantic notions of security χ, we studied compositionality w.r.t. each language construct, inferring from these syntactic criteria $\overline{\chi}$ automatically. Now it is time to have a closer look at the recursive clauses of $\overline{\chi}$ and see what they tell us about $\overline{\chi}$ independently of χ. First the easy cases:

- $\overline{\mathsf{mayT}}\ c$ holds iff c does not contain while loops.
- $\overline{\mathsf{discr}}\ c$ holds iff all atoms in c are $\sim$-preserving, a.k.a. high.
- $\overline{\mathsf{siso}}\ c$ holds iff all tests in c are $\sim$-compatible, a.k.a. low, and all atoms are $\sim$-compatible.

$\overline{\mathsf{siso}}\ c$ corresponds to a type system from Smith and Volpano [29] for scheduler independent security – this criterion is extremely harsh, forbidding high tests at If and While.

We now move to the more interesting cases. $\overline{\approx_{\mathsf{WT}}}\ c$ is equivalent to another, possibilistic type system from Smith and Volpano [29]. Here, high tests are allowed at If provided the branches are discreet, but are disallowed at While: $\overline{\approx_{\mathsf{WT}}}\ (\mathsf{While}\ tst\ c) =$

$$\begin{cases} \overline{\approx_{\mathsf{WT}}}\ c, & \text{if } \mathsf{cpt}\ tst \\ \overline{\mathsf{discr}}\ (\mathsf{While}\ tst\ c) \wedge \overline{\mathsf{mayT}}\ (\mathsf{While}\ tst\ c), & \text{otherwise} \end{cases} = \begin{cases} \overline{\approx_{\mathsf{WT}}}\ c, & \text{if } \mathsf{cpt}\ tst \\ \mathsf{False}, & \text{otherwise.} \end{cases}$$

The above harsh condition on While is the starting point of work by Boudol and Castellani in [5, 6], where a type system equivalent to $\overline{\approx_{01}}$ is introduced. $\overline{\approx_{01}}$ allows high tests for While provided the body of the While is discreet. This is possible because, unlike $\overline{\approx_{\mathsf{WT}}}$, $\overline{\approx_{01}}$ can fall back on $\overline{\mathsf{discr}}$:
$\overline{\approx_{01}}\ (\mathsf{While}\ tst\ c) = \overline{\mathsf{discr}}\,(\mathsf{While}\ tst\ c) \vee \overline{\mathsf{siso}}\,(\mathsf{While}\ tst\ c) = \overline{\mathsf{discr}}\,c \vee (\mathsf{cpt}\ tst \wedge \overline{\mathsf{siso}}\,c)$.
However, the price for this is a harsher clause for Seq (as we have seen, a limitation shared by all termination-insensitive notions). Indeed, $\overline{\approx_{\mathsf{WT}}}$ commutes smoothly with Seq as $\overline{\approx_{\mathsf{WT}}}\ (\mathsf{Seq}\ c_1\ c_2) = (\overline{\approx_{\mathsf{WT}}}\ c_1\ \wedge\ \overline{\approx_{\mathsf{WT}}}\ c_2)$, but $\overline{\approx_{01}}$ needs either $\overline{\mathsf{siso}}$ on the left or $\overline{\mathsf{discr}}$ on the right: $\overline{\approx_{01}}\ (\mathsf{Seq}\ c_1\ c_2) = (\overline{\mathsf{siso}}\ c_1\ \wedge\ \overline{\approx_{01}}\ c_2)\ \vee\ (\overline{\approx_{01}}\ c_1\ \wedge\ \overline{\mathsf{discr}}\ c_2)$.
Thus, $\overline{\approx_{01}}$ requires that either c_1 has only low tests, or c_2 has only high atoms. Hence, e.g., the command c_5 from Example 1 is accepted by $\overline{\approx_{\mathsf{WT}}}$, but rejected by $\overline{\approx_{01}}$.

An improvement of $\overline{\approx_{01}}$ that accepts c_5 also is proposed by Boudol in [4], where the idea is that, in the c_1 part of Seq $c_1\ c_2$, one should no longer restrict to low tests everywhere, but rather only in places that may affect termination (i.e., inside While loops). Interestingly, this condition on c_1 is the one imposed by $\overline{\approx_{\mathsf{WT}}}$, and therefore the approach of [4] can be seen as a carefully designed combination of $\overline{\approx_{\mathsf{WT}}}$ and $\overline{\approx_{01}}$. Remarkably, it turns out to be equivalent to $\overline{\approx_{\mathsf{W}}}$, whose Seq clause is: $\overline{\approx_{\mathsf{W}}}\ (\mathsf{Seq}\ c_1\ c_2) = (\overline{\approx_{\mathsf{WT}}}\ c_1\ \wedge\ \overline{\approx_{\mathsf{W}}}\ c_2)\ \vee\ (\overline{\approx_{\mathsf{W}}}\ c_1\ \wedge\ \overline{\mathsf{discr}}\ c_2)$.

In the above cited work, the soundness of the proposed type systems (results corresponding to Prop. 4) are given rather elaborate proofs, defining global bisimulation relations that involve multiple language constructs combined in ingenious and ad hoc ways. These proofs are often hard to understand and mechanize. Moreover, they are not exploiting the uniformities, commonalities and inter-dependencies of the various approaches. By contrast, our proof methodology is entirely local and uniform: we choose a language construct and a notion of security, and essentially do our best at proving (partial) compositionality. Then syntactic criteria follow automatically by our table-and-graph method. We were pleasantly surprised to find that this general method could capture such a variety of ad hoc results.

Finally, we discuss $\overline{\approx_{\mathsf{T}}}$, which is our own novel type-system-like criterion for noninterference. It turns out to be a natural extension of the original Volpano-Smith-Irvine typing of sequential programs [30], using the same clauses for the sequential part together with $\overline{\approx_{\mathsf{T}}}\ (\mathsf{Par}\ c_1\ c_2) = (\overline{\approx_{\mathsf{T}}}\ c_1\ \wedge\ \overline{\approx_{\mathsf{T}}}\ c_2)$. The reason why such a natural type system is absent from the literature is probably that its associated semantic notion of security, $\approx_{\mathsf{T}}$, was overlooked.

$\overline{\approx_{\mathsf{T}}}$ accepts the commands c_7 from Example 1 and $d \equiv c_7 \parallel l := 5$, while the most permissive criterion studied so far, $\overline{\approx_{\mathsf{W}}}$, rejects them. However, as discussed, the security property that $\overline{\approx_{\mathsf{T}}}$ guarantees, $\approx_{\mathsf{T}}$, is different from $\approx_{\mathsf{W}}$, the main restriction of $\approx_{\mathsf{T}}$ being that it only makes sense under the termination assumption. Thus, $\overline{\approx_{\mathsf{T}}}$ provides a useful guarantee for c_7 and d only in cases when the initial state s ensures termination, here, if it has $h \geq 0$. On the other hand, $\overline{\approx_{\mathsf{W}}}$ rejects d out of fear that its c_7 component may not terminate, which would yield the pipelining of the c_7 termination channel into a standard channel for d. Termination knowledge excludes such behavior, and this is where the new criterion $\overline{\approx_{\mathsf{T}}}$ is advantageous.

7 After-execution noninterference

The bisimilarity-based notions of security studied so far are rather complex, assuming an elaborate attacker model that interacts continuously with program execution—we call these *during-execution* noninterference. Often one is interested in a more tractable notion, as an input-to-output property, such as: a command is secure if, upon execution starting in indistinguishable states, the result states (after the command has finished executing) are again indistinguishable. We call such input-to-output properties *after-execution* noninterference.

So what are the after-execution guarantees of the various bisimilarities from §3? To answer this, we need some terminology. Given a configuration (c,s):

- A *finite execution trace starting in* (c,s) (*finite* (c,s)*-trace* for short) is a finite sequence of the form $(c_0,s_0),(c_1,s_1),\ldots,(c_{n-1},s_{n-1}),s_n$ (consisting of a number of configurations followed by a state) such that $(c_0,s_0)=(c,s)$, $(c_i,s_i)\rightarrow_{\mathsf{c}}(c_{i+1},s_{i+1})$ for all $i<n-1$, and $(c_{n-1},s_{n-1})\rightarrow_{\mathsf{T}}s_n$. Then n is said to be the *length* of the trace and s_n the *final state* of the trace.
- An *infinite execution trace starting in* (c,s) (*infinite* (c,s)*-trace* for short) is an infinite sequence of the form $(c_0,s_0),(c_1,s_1),\ldots$ (consisting of configurations only) such that $(c_0,s_0)=(c,s)$ and $\forall i.\ (c_i,s_i)\rightarrow_{\mathsf{c}}(c_{i+1},s_{i+1})$.

Given a finite (c,s)-trace tr, $\mathsf{length}(tr)$ denotes its length and $\mathsf{fstate}(tr)$ denotes its final state. Thus, finite (c,s)-traces represent the terminating computations starting in (c,s), and infinite (c,s)-traces the divergent computations starting in (c,s). Note that (c,s) "must terminate" (as defined in §5) iff there exist no infinite (c,s)-traces. It is not hard to prove the following about the termination-sensitive bisimilarities:

Prop 6 (1) If $c\approx_{\mathsf{s}}c$ and $s\sim t$, then, for every finite (c,s)-trace tr, there exists a finite (c,t)-trace tr' with $\mathsf{fstate}(tr')\sim\mathsf{fstate}(tr)$ and $\mathsf{length}(tr')=\mathsf{length}(tr)$.
(2) If $c\approx_{01}c$ and $s\sim t$, then, for every finite (c,s)-trace tr, there exists a finite (c,t)-trace tr' with $\mathsf{fstate}(tr')\sim\mathsf{fstate}(tr)$ and $\mathsf{length}(tr')\leq\mathsf{length}(tr)$.
(3) If $c\approx_{\mathsf{WT}}c$ and $s\sim t$, then, for every finite (c,s)-trace tr, there exists a finite (c,t)-trace tr' with $\mathsf{fstate}(tr')\sim\mathsf{fstate}(tr)$.

Thus, for self strongly bisimilar commands, terminating executions starting in indistinguishable states have, up to indistinguishability, the same outcomes, obtained in the same amount of time—this means both standard (low data) channels and timing channels are secure here. For self weakly T-bisimilar commands, again the outcomes are the same up to indistinguishability, but timing channels are no longer secured. As usual, 01T-bisimilarity lies in between—there is a time guarantee, but weaker than perfect synchronization.

Now, turning to the termination-insensitive notions, during-execution security faces the difficulty that here terminating executions need not be matched by terminating executions. However, we can still prove a termination-conditioned result:

Prop 7 If $\forall s'.\ \mathsf{mustT}\ (c,s')$, then Prop. 6(3) holds with $\approx_{01}$ or $\approx_{\mathsf{W}}$ substituted for $\approx_{\mathsf{WT}}$.

Thus, in the termination-insensitive case, the after-execution distinction between 01- and weak bisimilarity vanishes. As for the after-execution guarantee of our termination-sensitive security notion $\approx_{\mathsf{T}}$ from §5, it is weaker than that of $\approx_{\mathsf{WT}}$ (Prop. 6(3)), but stronger than that of $\approx_{01}$ and $\approx_{\mathsf{W}}$ (Prop. 7):

Prop 8 If $\mathsf{mustT}\ (c,s)$, then Prop. 6(3) holds with $\approx_{\mathsf{T}}$ substituted for $\approx_{\mathsf{WT}}$.

8 Conclusions and more related work

This paper was concerned with systematizing and comparing existing type-system based noninterference results from the literature. As a technical tool, we have introduced a compositionality "table-and-graph" technique able to capture such results in a uniform way. The study also suggested a novel, suitably compositional, notion, the termination-interactive bisimilarity $\approx_{\mathrm{T}}$.

Our approach has important precursors in the literature. Thus, [25] makes a strong case for compositionality, and illustrates how it can be used to extend to concurrency a noninterference result [1] in the style of Volpano and Smith. However, [25] does not pursue this idea systematically or devise a general technique as we do in this paper. Moreover, our bisimilarity-based treatment employs insight from process algebra [16] in general and from process algebra approaches to noninterference [8] in particular. In system-based security, [9, 11, 15] provide general frameworks for trace-based system security, the last two having a special focus on compositionality and the first also incorporating probabilistic systems.

Themes missing from the compositionality framework discussed in this paper are probabilistic noninterference [13,26–28], dynamic thread creation [13,25,31] and scheduler independence [6, 13, 25, 31], known to be particularly problematic w.r.t. noninterference. Incorporating some of these features in our compositional setting is a goal for future research.

Another exciting future direction is a framework for proving concurrent noninterference by a combination of automated and interactive methods along the lines of approaches going beyond type systems [2, 7, 12]. This would follow a rely-guarantee paradigm [10], with information about the environment made available to individual threads by suitably relaxing interactivity. A step towards this direction is made by our termination-interactive bisimilarity $\approx_{\mathrm{T}}$, where such context information is termination, but could in principle be any liveness property.

Acknowledgements. We are grateful to Jasmin Blanchette for lots of suggestions hat have significantly improved the presentation of this paper, to Benedict Nordhoff and Peter Lammich for noticing various technical typos, and to the anonymous reviewers for useful comments.

References

1. J. Agat. Transforming out timing leaks. In *POPL*, pages 40–53, 2000.
2. G. Barthe, P. R. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *IEEE Computer Security Foundations Workshop*, pages 100–114, 2004.
3. G. Barthe and L. P. Nieto. Formally verifying information flow type systems for concurrent and thread systems. In *FMSE*, pages 13–22, 2004.
4. G. Boudol. On typing information flow. In *ICTAC*, pages 366–380, 2005.
5. G. Boudol and I. Castellani. Noninterference for concurrent programs. In *ICALP*, pages 382–395, 2001.
6. G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1-2):109–130, 2002.
7. Á. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In *SPC*, pages 193–209, 2005.

8. R. Focardi and R. Gorrieri. Classification of security properties (part i: Information flow). In *FOSAD*, pages 331–396, 2000.
9. J. Y. Halpern and K. R. O'Neill. Secrecy in multiagent systems. *ACM Trans. Inf. Syst. Secur.*, 12(1), 2008.
10. C. B. Jones. Specification and design of (parallel) programs. In *IFIP Congress'83*, pages 321–332, 1983.
11. H. Mantel. On the composition of secure systems. In *IEEE Symposium on Security and Privacy*, pages 88–, 2002.
12. H. Mantel, D. Sands, and H. Sudbrock. Assumptions and guarantees for compositional noninterference. In *CSF'11*, pages 218–232, Cernay-la-Ville, France, 2011.
13. H. Mantel and H. Sudbrock. Flexible scheduler-independent security. In *ESORICS*, pages 116–133, 2010.
14. H. Mantel, H. Sudbrock, and T. Kraußer. Combining different proof techniques for verifying information flow security. In *LOPSTR*, volume 4407 of *LNCS*, pages 94–110. 2007.
15. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. pages 79–93, May 1994.
16. R. Milner. *Communication and concurrency*. Prentice Hall, 1989.
17. G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.
18. A. Popescu and J. Hölzl. Possibilistic noninterference formalized in Isabelle/HOL. *Archive for Formal Proofs*, ?, 2012. urlhttp://1234.
19. A. Russo, J. Hughes, D. Naumann, and A. Sabelfeld. Closing internal timing channels by transformation. In *ASIAN 2006*, volume 4435 of *LNCS*, pages 120–135. 2007.
20. A. Russo and A. Sabelfeld. Security for multithreaded programs under cooperative scheduling. In *Perspectives of Systems Informatics*, volume 4378 of *LNCS*, pages 474–480. 2007.
21. A. Sabelfeld. The impact of synchronisation on secure information flow in concurrent programs. In *Perspectives of System Informatics*, volume 2244 of *LNCS*, pages 225–239. 2001.
22. A. Sabelfeld. Confidentiality for multithreaded programs via bisimulation. In *International Conference on Perspectives of System Informatics*, LNCS, pages 260–273, 2003.
23. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.
24. A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. In *European Symposium on Programming*, volume 1576 of *LNCS*, pages 40–58, 1999.
25. A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *IEEE Computer Security Foundations Workshop*, pages 200–214, 2000.
26. G. Smith. A new type system for secure information flow. In *IEEE Computer Security Foundations Workshop*, pages 115–125, 2001.
27. G. Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *IEEE Computer Security Foundations Workshop*, pages 3–13, 2003.
28. G. Smith. Improved typings for probabilistic noninterference in a multi-threaded language. *Journal of Computer Security*, 14(6):591–623, 2006.
29. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *ACM Symposium on Principles of Programming Languages*, pages 355–364, 1998.
30. 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.
31. S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *IEEE Computer Security Foundations Workshop*, pages 29–43, 2003.