Formal Verification of Language-Based Concurrent Noninterference

Andrei Popescu
Technische Universität München and Institute of Mathematics Simion Stoilow of the Romanian Academy
Johannes Hölzl
Technische Universität München
and
Tobias Nipkow
Technische Universität München

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 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 syntactic criterion. 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 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 [3–7, 20, 32–34, 37, 42, 47], many of them surveyed in [36]. There is usually a tradeoff between the strength of a security property and the harshness of the

This is an extended version of the conference paper [31]; it additionally includes more detailed explanations of the discussed concepts and techniques, proof sketches, a presentation of the Isabelle formalization, and addresses more related work. The work reported here has been supported by the DFG project Ni 491/13–2 (part of the DFG priority program Reliably Secure Software Systems–RS3) and by NI 491/15-1.

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 simpler proofs. As a preview of the kind of properties we 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 [38]).

(4) A relaxation of strong bisimilarity with lock-step synchronization replaced by *01-bisimilarity* (simply called *bisimilarity* in [6]), 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* [22] (used in [5,42] 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 $\|$ is parallel composition with interleaving semantics, l is a low variable and h, h' are high variables:

—c_0: $h := 0$
—c_1: if $l = 0$ then $h := 1$ else $l := 2$
—c_2: if $h = 0$ then $h := 1$ else $h := 2$
—c_3: if $h = 0$ then $h := 1$; $h := 2$ else $h := 3$

—c_4: $l := 4 \ ; \ c_3$
—c_5: $c_3 \ ; \ l := 4$
—c_6: $c_1 \parallel c_4 \parallel c_5$
—c_7: $c_1 \parallel c_2$
—c_8: $l := h$
—c_9: $h' := 0$; while $h > 0$ do $\{h := h - 1 \ ; \ h' := h' + 1\}$; $l := 4$
—c_{10}: $c_9 \parallel l := 1 \parallel h := 0 \parallel h := h + 1$

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 because, one of the branches of its prefix c_3 being faster, it reaches the visible step $l := 4$ at a time when this step is not immediately available on the alternative branch; but c_5 is self weakly bisimilar, since weak bisimilarity allows the alternative branch to catch up using multiple discreet steps.

c_6 is a pool consisting of three threads running in parallel: a self isomorphic one c_1, a self 01-bisimilar one c_4 and a self weakly bisimilar one c_5. One would hope, in a compositional framework, that the pool satisfies the weakest of the security properties of its threads, here, weak self bisimilarity. As we shall see, this is indeed the case if the above informal notions of security are formalized in an *interactive* manner, taking into consideration the possibility that the environment too may change the program state. But how about c_7? It is a pool of a self isomorphic thread c_1 and a discreet thread c_2. Since these two security notions are not comparable, we should have c_7 secure according to the strongest available security property that is weaker than both—amongst our considered properties, this is self 01-bisimilarity.

c_8 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_9 secure, since it has the same effect as the program $h' := h \ ; \ l := 4$. However, whether or not we should deem c_9 secure *when placed in parallel with other threads* depends on the assumption we make on these threads—e.g., are they allowed to increase h, thus preventing termination of c_9? For instance, the pool c_{10} containing c_9 is harmless, since:

—the thread $l := 1$ does not affect the termination of c_9, not writing the counter h;
—the thread $h := 0$ helps the termination of c_9 by setting the counter to an exit value;
—the thread $h := h + 1$, although it *delays* the termination of c_9, does not *prevent* its termination.

Later we shall phrase this termination-conditional notion of security in a compositional manner.

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 (Section 2). Then we systematize and compare bisimilarity-based notions from

the literature (Section 3). A formal study of the compositionality of, and of the implications between, these notions (Section 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 (Section 5). We then look at existing work on security type systems in the light of our analysis (Section 6)—the aforementioned simple proof technique turns out quite insightful, capturing these type system criteria uniformly. Our novel security notion from Section 5 yields a more permissive syntactic criterion than the existing ones, but the result targets only terminating programs. We also discuss end-to-end security aspects of the studied bisimilarity-based notions (Section 7). Finally, we present the mechanization of our constructions and results in Isabelle/HOL (Section 8).

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:

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

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 $c_1\ ;\ c_2$. $\mathsf{If}\ tst\ c_1\ c_2$ is the conditional, written in concrete syntax as $\mathsf{if}\ tst\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$, where *tst* ranges over an unspecified set **test** of tests. Standard examples of tests are Boolean expressions such as $x = y$. $\mathsf{While}\ tst\ c$ is the usual while loop, in concrete syntax, $\mathsf{while}\ tst\ \mathsf{do}\ c$. $\mathsf{Par}\ c_1\ c_2$ 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, $\mathsf{aexec} : \mathbf{atom} \to \mathbf{state} \to \mathbf{state}$;

—an evaluation function for the tests, $\mathsf{tval} : \mathbf{test} \to \mathbf{state} \to \mathbf{bool}$.

Then we define a standard small-step semantics [27] as a pair of inductive predicates $\to_\mathsf{T} : (\mathbf{com} \times \mathbf{state}) \to \mathbf{state}$ and $\to_\mathsf{C} : (\mathbf{com} \times \mathbf{state}) \to (\mathbf{com} \times \mathbf{state})$ (where the subscripts T and C stand for "termination" and "continuation") specified in Figure 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

The freeness of the command constructors in conjunction with the inductive definitions of $\to_{\mathsf{C}}$ and $\to_{\mathsf{T}}$ provide us with the obvious inversion rules. E.g., the inversion rule for $\to_{\mathsf{C}}$ w.r.t. Seq is the following: If $(\mathsf{Seq}\ c_1\ c_2,s)\to_{\mathsf{C}}(c',s')$, then one of the following holds:

—either $(c_1,s)\to_{\mathsf{T}} s'$ and $c'=c_2$,
—or there exists c_1' such that $(c_1,s)\to_{\mathsf{C}}(c_1',s')$ and $c'=\mathsf{Seq}\ c_1'\ c_2$.

Similar, but slightly more complex rules can be proved for the "star" relations, e.g., for $\to_{\mathsf{C}}^*$ w.r.t. Seq: If $(\mathsf{Seq}\ c_1\ c_2,s)\to_{\mathsf{C}}^*(c',s')$, then one of the following holds:

—either there exist c_1' and s'' such that $(c_1,s)\to_{\mathsf{T}} s''$ and $(c_2,s'')\to_{\mathsf{C}}^*(c',s')$.
—or there exists c_1' such that $(c_1,s)\to_{\mathsf{C}}^*(c_1',s')$ and $c'=\mathsf{Seq}\ c_1'\ c_2$.

Another useful fact will be the following progress property of the operational semantics:

PROPOSITION 1. For all c and s, either there exists s' such that $(c,s)\to_{\mathsf{T}} s'$ or there exist c' and s' such that $(c,s)\to_{\mathsf{C}}(c',s')$.

PROOF. By a straightforward structural induction on c. □

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." The only assumption we make about $\sim$ is that it is an equivalence relation. While the results of this paper work abstractly for any such $\sim$, in the literature on language-based security it is often the case that one considers a particular $\sim$, which we shall also use in our examples and informal discussions.

EXAMPLE 2. 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, $s \sim t \equiv \forall x \in \mathbf{var}.\ \mathsf{sec}\ x = \mathsf{lo} \Longrightarrow s\ x = t\ x$. [1]

3.1 Security Notions Given by Unary Predicates

We start with some very simple notions of security, describable directly as unary predicates. Namely, we define the following predicates on commands *coinductively as greatest fixed points*, i.e., as the *weakest*[2] 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 $\mathsf{discr}\ c \equiv$
$(\forall s\ c'\ s'.\ (c,s) \rightarrow_{\mathsf{c}} (c',s') \Longrightarrow s \sim s' \wedge \mathsf{discr}\ c') \wedge (\forall s\ s'.\ (c,s) \rightarrow_{\mathsf{T}} s' \Longrightarrow s \sim s')$

Note that here $\equiv$ does not introduce a definition in the usual sense, but rather expresses an equality that, together with the greatest fixed point assumption, identifies uniquely the introduced concept. This will be the case with all the coinductive definitions throughout this paper.

According to the definition of siso, a command c is self isomorphic iff the following hold for all indistinguishable states s and t:

—Any single-step continuation c' of c available from state s is also available from t, yielding indistinguishable states s' and t'. (This is stated in the first conjunct from the definition of siso.)

—Any single-step terminating state s' of c available from state s has an indistinguishable counterpart, t', available from t. (This is stated in the second conjunct.)

—The above 2 facts hold for all single-step continuations c' of c, and for all single-step continuations c'' of c', *and so on, indefinitely*. (This is stated in the third conjunct, together with the maximal interpretation of siso, as greatest fixed point.)

3.2 Interactiveness and Possibilism

The coinductive definition of self isomorphism expresses that the command execution 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 execution changes the indistinguishability class, regardless of what that class has become through potential environment action.

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

[1] In the mechanization described in Section 8, we consider the more general case of *multilevel security*, via a lattice of security levels—however, this brings neither much additional difficulty, nor much additional insight, so here we focus on this 2-level lattice, with $\mathsf{lo} < \mathsf{hi}$.

[2] The conference version of this paper [31] erroneously used the superlative "strongest" instead of "weakest".

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 considered 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 [35, 38]) 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. In other words, interactivity ensures compositionality w.r.t. parallel composition. As a consequence, most of the security notions discussed in this paper will be interactive.

Another prominent aspect of the notions discussed in this paper is their *possibilistic* nature, indicated by existential quantification on the right of some of their defining clauses (for self-isomorphism, $\exists t'$): given two indistinguishable states, for any (multi-)step taken from one state, *there exists* a matching (multi-)step taken from the other so that the results are again indistiguishable.

We shall also need the following notion of termination possibility at each point during execution, via the coinductively defined predicate mayT (read "may terminate"):

$$\mathsf{mayT}\, c \equiv \forall s\, c'\, s'.(c,s)\rightarrow_{\mathsf{c}}(c',s') \Longrightarrow (\exists s''.(c',s')\rightarrow_{\mathsf{T}}^{*} s'') \wedge \mathsf{mayT}\, c'.$$

Thus, may-termination means: for any continuation of the command and any given state, there is a possible terminating execution.

3.3 Security Notions Given by Bisimilarities

Self isomorphism and discreetness were expressible as unary predicates. However, interesting noninterference properties may require binary relations. In order to illustrate this, let us assume we wish to express that c is secure in that the executions of c are (multi)stepwise equivalent if started in indistinguishable states. Suppose 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)stepwise 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 bisimilarity.

Parenthesis. Before doing this, we give some context for readers unfamiliar with process algebra. Suppose we have a single-step transition relation between processes. Then two processes p and q are called *bisimilar* if the following hold:

(1) for all continuations p' of p, there exists a continuation q' of q such that p' and q' are again bisimilar;
(2) and vice versa (with p and q switched).

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

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

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

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

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

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

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

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

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

Fig. 2: Matchers

We can think of these conditions as an infinitary two-player game between an "attacker" who tries to distinguish two processes p and q (and thus disprove bisimilarity) and a "defender" who tries to prevent this. At each moment, the state of the game consists of a pair (p,q). The attacker picks a process, say, p, and a transition from it, say, $p \rightarrow p'$, the new state becoming (p',q). Then the defender has to find a transition from q, say $q \rightarrow q'$, the new state becoming (p',q'), and so on. The attacker wins if, upon his turn, the defender can no longer move; otherwise, i.e., if (upon his turn) the attacker can no longer move or if the defender manages to stay in the game indefinitely, then the defender wins. Two processes p and q are bisimilar iff the attacker has no winning strategy. (Here, we are in a very simple setting where two processes are only distinguishable by means of termination, i.e., if, at some point, one cannot move while the other can. In general, however, one assumes further means for the attacker to inspect processes, e.g., an observer function on their current state.)

Technically, the above amounts to defining bisimilarity as the *weakest symmetric relation* θ for which the following holds for all p and q such that $\theta\ p\ q$: for all p' such that $p \rightarrow p'$, there exists q' such that $q \rightarrow q'$ and $\theta\ p'\ q'$. If we extract the core of this characterization into a predicate match defined as $\mathsf{match}\ \theta\ p\ q \equiv \forall p'.\ p \rightarrow p' \Longrightarrow (\exists q'.\ q \rightarrow q' \wedge \theta\ p'\ q')$, then bisimilarity $\approx$ can be defined as the weakest symmetric relation θ for which $\theta\ p\ q$ is equivalent to $\mathsf{match}\ \theta\ p\ q$ for all processes p,q. We write this briefly as the fixed point definition $\approx p\ q \equiv \mathsf{match} \approx p\ q$, interpreted maximally in the space of symmetric relations.

The advantage of employing the auxiliary concept of a matcher is of course the technical ease of varying the notion of bisimilarity. For instance, if we want weak bisimilarity (i.e., one step matched by multiple steps), we modify the definition of match to use $\rightarrow^*$, the transitive closure of $\rightarrow$, on the right-hand side of the implication: $\mathsf{match}\ \theta\ p\ q \equiv \forall p'.\ p \rightarrow p' \Longrightarrow (\exists q'.\ q \rightarrow^* q' \wedge \theta\ p'\ q')$. *End of parenthesis.*

Our relevant matchers are defined in Figure 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 *weakest 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 $c \approx_{01} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{01}}\ (\approx_{01})\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{01}}\ c\ d$

—*Termination-sensitive 01-bisimilarity* (*01T-bisimilarity*), $\approx_{01\mathsf{T}}$, by $c \approx_{01\mathsf{T}} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{01C}}\ (\approx_{01\mathsf{T}})\ c\ d \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d$

—*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 $c \approx_{\mathsf{WT}} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{MC}}\ (\approx_{\mathsf{W}})\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{MT}}\ c\ d$

To be more precise, the above relations are defined using the Knaster-Tarski theorem, stating that every monotonic function $P : L \rightarrow L$ on a complete lattice $(L, \leq)$ has a greatest fixpoint $\mathsf{gfp}\ P \in L$, i.e., such that $P\ (\mathsf{gfp}\ P) = \mathsf{gfp}\ P$ and $\forall x \in L.\ P\ x = x \Longrightarrow x \leq \mathsf{gfp}\ x$. In fact, $\mathsf{gfp}\ P$ is also the greatest postfixpoint, i.e., $\forall x \in L.\ x \leq P\ x \Longrightarrow x \leq \mathsf{gfp}\ x$. This forms the basis of the coinduction principle: to prove $x \leq \mathsf{gfp}\ P$, it suffices to prove $x \leq P\ x$; or, in a slightly more convenient form: to prove $x \leq \mathsf{gfp}\ P$, it suffices to prove $x \leq P\ (x\ \vee\ \mathsf{gfp}\ x)$, where $\vee$ is lattice (binary) supremum. We call an x such that $x \leq P\ (x\ \vee\ \mathsf{gfp}\ x)$ a *generalized postfixpoint*. Thus, the coinduction principle, which will be heavily employed

in this paper, is about proving that certain items are generalized postfixpoints of certain monotonic functions.

In our case, L is the set of symmetric binary relations $\theta : \mathbf{com} \to \mathbf{com} \to \mathbf{bool}$ and $\leq$ is inclusion (point-wise implication), and the various operators P are given by combinations of matchers. For example:

— $\approx_{\mathsf{S}}$ is the gfp of $\lambda\ \theta.\ \lambda\ c\ d.\ \mathsf{match}^{\mathsf{C}}_{\mathsf{C}}\ \theta\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d$
— $\approx_{01\mathsf{T}}$ is the gfp of $\lambda\ \theta.\ \lambda\ c\ d.\ \mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ \theta\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d$

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.

3.4 Attacker Models

Next 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[3] 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. As usual, the attacker does not know the initial values of the high variables and is trying to infer information about them—security thus refers to the impossibility of the attacker to infer anything by running the program and performing experiments on its states, perhaps also while the program is still running. Security of a command c is therefore expressed as: For what the attacker knows, the initial values of the high variables could have been any. That is, provided the initial state is, say, s, any indistinguishable state t (i.e., for the standard choice of $\sim$ from Example 2, a state t agreeing with s on the low variables, but not necessarily on the high ones) would have exhibited the same attacker-observable behavior. If we also factor in interactivity (meaning that, due to the intervention of the environment, fresh indistinguishable states s and t may replace the existing ones at any point during the execution), we obtain the following property: By keeping experimenting on the continuations of c with different indistinguishable states s and t, the attacker cannot discover any differences in the resulting states s' and t' or (in the termination-sensitive cases) in the fact that execution terminates. The latter is (self) bisimilarity.

The various notions of bisimilarity differ in further assumptions on how precisely is the attacker assumed to be able to experiment on the running program and its states. 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, [4] 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

[3] This is a worse-case-scenario assumption. E.g., in Trojan Horse attacks, the attacker is assumed to own the program code.
[4] Here and elsewhere, by the attacker "inspecting the state" we mean "inspecting the *low* part of the state".

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.

3.5 Hierarchy of the Security Notions

The following proposition orders the different notions of self bisimilarity according to their strength:

PROPOSITION 2. The implications in Figure 3 hold.

PROOF. We distinguish two types of stated facts:

1) Implications between genuinely unary predicates and other predicates:

—$\mathsf{discr}\ c \Longrightarrow c \approx_{01} c$

—$\mathsf{discr}\ c \wedge \mathsf{mayT}\ c \Longrightarrow c \approx_{\mathsf{WT}} c$

—$\mathsf{siso}\ c \Longrightarrow c \approx_{\mathsf{S}} c$

After suitably rephrasing them, e.g., $\mathsf{discr}\ c \wedge c = d \Longrightarrow c \approx_{01} d$, these facts follow by routine coinduction on the right-hand side predicates, $\approx_{01}$ and $\approx_{\mathsf{S}}$.

2) Implications between predicates stemming from binary relations—for these we prove the more general binary versions:

—$c \approx_{\mathsf{S}} d \Longrightarrow c \approx_{01\mathsf{T}} d$

—$c \approx_{01\mathsf{T}} d \Longrightarrow c \approx_{01} d$

—etc.

Instead of doing coinductive proofs here, we invoke a general result stating that greatest fixed points are monotonic: If $P, Q : L \to L$ are monotonic functions on a complete lattice $(L, \leq)$ and $P \leq Q$ (that is to say, $\forall x \in L.\ P\,x \leq Q\,x$), then $\mathsf{gfp}\ P \leq \mathsf{gfp}\ Q$. In our case, $P \leq Q$ amounts to implications between conjunctions of matchers:

—$\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}\ \theta\ c\ d \wedge \mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{C}}_{01}\ \theta\ c\ d \wedge \mathsf{match}^{\mathsf{T}}_{01}\ c\ d$

—$\mathsf{match}^{\mathsf{C}}_{01}\ \theta\ c\ d \wedge \mathsf{match}^{\mathsf{T}}_{01}\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ \theta\ c\ d \wedge \mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d$

—etc.

These follow from implications between matchers:

—$\mathsf{match}^{\mathsf{C}}_{\mathsf{C}}\ \theta\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ \theta\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{C}}_{\mathsf{MC}}\ \theta\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{C}}_{\mathsf{M}}\ \theta\ c\ d$

—$\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ \theta\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{C}}_{01}\ \theta\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{C}}_{\mathsf{M}}\ \theta\ c\ d$

—$\mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{T}}_{01}\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{T}}_{\mathsf{M}}\ c\ d$

—$\mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{T}}_{\mathsf{MT}}\ c\ d \Longrightarrow \mathsf{match}^{\mathsf{T}}_{\mathsf{M}}\ c\ d$

The last implications are trivial. □

Note that discreetness trivially implies self 01-bisimilarity; however, it does not imply self 01T-bisimilarity, since intuitively it does not exclude that, starting in different indistinguishable states, one execution terminates while the other has no possibility to terminate immediately. In fact, the other execution may not have the possibility to ever terminate, which shows that discreetness does not imply weak T-bisimilarity either. The last counterexample is excluded by may-termination, a property that postulates precisely that there is always a way to terminate; in fact, in the presence of may-termination, discreetness does imply self weak T-bisimilarity. This will be the key to understanding compositionally the syntactic criteria à la Smith and Volpano, which accept discreet non-self-isomorphic programs only if they have finite behavior. (See also the discussion on page 20.)

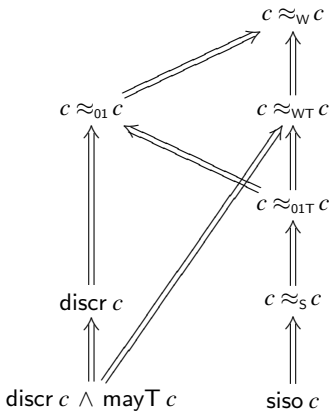

Figure 3: Implications between security notions

	1	2	3	4	5
1	c	mayT c	discr c	$\varphi\ c$	$\psi\ c$
2	*atm*	True	pres *atm*	cpt *atm*	cpt *atm*
3	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
4	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* $\psi\ c_1$ $\psi\ c_2$
5	While *tst* d	False	discr d	cpt *tst* $\varphi\ d$	False
6	Par $c_1\ c_2$	mayT c_1 mayT c_2	discr c_1 discr c_2	$\varphi\ c_1$ $\varphi\ c_2$	$\psi\ c_1$ $\psi\ c_2$

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

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

Figure 4: Compositionality table

Example 1 already illustrates most of the above bisimilarities. Here are some further illustrations that also take Proposition 2 into account (using the Example 1 notations).

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 transit in one step (according to the operational semantics) to either $d \equiv h := 1\ ;\ h := 2$ 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; then, by the definition of bisimilarity, c_3 is not self-01T-bisimilar either.
(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, namely, self isomorphism, discreetness and strong bisimilarity, are too harsh requirements, but, as we shall see, turn out as useful auxiliaries.) Smith and Volpano [42] focus on termination-sensitive weak bisimilarity. On the other hand, Boudol and Castellani [6, 7] prefer termination-insensitive 01-bisimilarity, while later Boudol [5] 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 *condition* part of the contract (while the bisimilarities 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.

The following definitions of $\sim$- preservation and compatibility for tests and atoms will represent basic building blocks in our compositionality analysis. All atoms will be assumed at least $\sim$-compatible and occasionally tests will be assumed $\sim$-compatible as side-conditions for compositionality w.r.t. If and While.

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, the above notions can be understood as follows. 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). For tests, $\sim$-compatibility means no dependence on high variables.

The next proposition states various compositionality results, schematically represented in Figure 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 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 $c \approx_{\mathsf{W}} c$.

PROPOSITION 3. The compositionality facts stated in Figure 4 hold.

PROOF. As before, for statements involving binary relations, we prove the more general form that does not assume the arguments equal. Thus, e.g., for $\psi = \approx_{01}$ in row 3 column 5, we prove facts (1) and (2) below; for $\varphi = \approx_{\mathsf{s}}$ in row 5 column 4 we prove fact (3) below:

(1) $c_1 \approx_{01\mathsf{T}} d_1 \wedge c_2 \approx_{01} d_2 \Longrightarrow \mathsf{Seq}\ c_1\ c_2 \approx_{01} \mathsf{Seq}\ d_1\ d_2$

(2) $c_1 \approx_{01} d_1 \wedge \mathsf{discr}\ c_2 \Longrightarrow \mathsf{Seq}\ c_1\ c_2 \approx_{01} \mathsf{Seq}\ d_1\ c_2$

(3) $\mathsf{cpt}\ tst \wedge c \approx_{\mathsf{s}} d \Longrightarrow \mathsf{While}\ tst\ c \approx_{\mathsf{s}} \mathsf{While}\ tst\ d$

All the proofs are performed by coinduction on the right-hand side predicate, again, after suitable rephrasing, e.g.:

(1) $(\exists c_1\ d_1\ c_2\ d_2.\ c = \mathsf{Seq}\ c_1\ c_2\ \wedge\ d = \mathsf{Seq}\ d_1\ d_2\ \wedge\ c_1 \approx_{01\mathsf{T}} d_1\ \wedge\ c_2 \approx_{01} d_2) \Longrightarrow c \approx_{01} d$
is proved by coinduction on $\approx_{01}$

(2) $(\exists c_1\ d_1\ c_2.\ c = \mathsf{Seq}\ c_1\ c_2\ \wedge\ d = \mathsf{Seq}\ d_1\ c_2\ \wedge\ c_1 \approx_{01} d_1\ \wedge\ \mathsf{discr}\ c_2) \Longrightarrow c \approx_{01} d$
is proved by coinduction on $\approx_{01}$

(3) Assuming $\mathsf{cpt}\ tst$, $(\exists c_1\ d_1.\ c = \mathsf{While}\ tst\ c_1\ \wedge\ d = \mathsf{While}\ tst\ d_1 \wedge c_1 \approx_{\mathsf{S}} d_1) \Longrightarrow c \approx_{\mathsf{S}} d$
is proved by coinduction on $\approx_{\mathsf{S}}$

The coinduction proofs proceed rather straightforwardly, albeit very tediously—they are similar to the bisimilarity-preservation proofs for process algebra operators [2, 22]. One needs to prove that the left-hand side of the implication (or occasionally, a slightly stronger relation) is a generalized postfixpoint of the operator defining the right-hand side (i.e., a bisimulation of a certain kind)—this process involves considering the operational semantics back and forth, that is, using inversion rules followed by the direct rules used in the definition of the semantics.

E.g., for (1), we define $\theta\ c\ d \equiv \exists c_1\ d_1\ c_2\ d_2.\ c = \mathsf{Seq}\ c_1\ c_2 \wedge d = \mathsf{Seq}\ d_1\ d_2 \wedge c_1 \approx_{01\mathsf{T}} d_1 \wedge c_2 \approx_{01} d_2$. Recall from page 10 that the complete lattice L used to define our notions of security is the set of symmetric binary relations on commands. We first show that $\theta \in L$, i.e., that θ is symmetric—this is immediate. It remains to show that $\theta \leq \approx_{01}$. Applying coinduction, it suffices to show that θ is a generalized postfixpoint of P, i.e., $\theta \leq P\ (\theta \vee \approx_{01})$, where $P : L \to L$ is the operator that defines $\approx_{01}$ as its greatest fixpoint. By the definition of P, this amounts to assuming that (A) $\theta\ c\ d$ holds, and showing that (B) $\mathsf{match}^{\mathsf{C}}_{01}\ (\theta \vee \approx_{01})\ c\ d$ and (C) $\mathsf{match}^{\mathsf{T}}_{01}\ c\ d$ hold.

We shall prove goal (B). ((C) follows by a similar reasoning.) Unfolding the definition of $\mathsf{match}^{\mathsf{C}}_{01}$, we reduce (B) to the following: We fix s,t,c',s' and assume (E) $s \sim t$ and (F) $(c,s) \to_{\mathsf{c}} (c',s')$. We need to show that one of the following three facts hold:
(G1) $\exists d'\ t'.\ (d,t) \to_{\mathsf{c}} (d',t') \wedge s' \sim t' \wedge (\theta\ c'\ d' \vee c' \approx_{01} d')$
(G2) $s' \sim t \wedge (\theta\ c'\ d \vee c' \approx_{01} d)$
(G3) $\exists t'.\ (d,t) \to_{\mathsf{T}} t' \wedge s' \sim t' \wedge \mathsf{discr}\ c'$
From (A), we obtain c_1, d_1, c_2, d_2 such that
(A1) $c = \mathsf{Seq}\ c_1\ c_2$, (A2) $d = \mathsf{Seq}\ d_1\ d_2$, (A3) $c_1 \approx_{01\mathsf{T}} d_1$, and (A4) $c_2 \approx_{01} d_2$.
From (A3) and the property of $\approx_{01\mathsf{T}}$ of being a fixpoint , we obtain
(A31) $\mathsf{match}^{\mathsf{C}}_{01\mathsf{C}}\ (\approx_{01\mathsf{T}})\ c_1\ d_1$ and (A32) $\mathsf{match}^{\mathsf{T}}_{\mathsf{T}}\ c_1\ d_1$
From (A1) and (F), using the inversion rule for $\to_{\mathsf{c}}$ w.r.t. Seq, we split in 2 cases:

Case 1: $c' = c_2$ and $(c_1,s) \to_{\mathsf{T}} s'$. With (A32) and (E), we obtain t' such that $s' \sim t'$ and $(d_1,t) \to_{\mathsf{T}} t'$. Hence, from (A2) and the (direct) rules for $\to_{\mathsf{c}}$, $(d,t) \to_{\mathsf{c}} (d_2,t')$. Thus, taking $d' = d_2$, with (A4) we obtain (G1) (by satisfying the $c' \approx_{01} d'$ part of the disjunction), as desired.

Case 2: c' has the form $\mathsf{Seq}\ c'_1\ c_2$ for some c'_1 such that $(c_1,s) \to_{\mathsf{c}} (c'_1,s')$. With (A31) and (E), we split in 2 subcases:

- Case 2.1: There exist d'_1 and t' such that $(d_1,t) \to_{\mathsf{c}} (d'_1,t')$, $s' \sim t'$, and $c'_1 \approx_{01\mathsf{T}} d'_1$. By the definition of $\to_{\mathsf{c}}$, we have $(\mathsf{Seq}\ d_1\ d_2,t) \to_{\mathsf{c}} (\mathsf{Seq}\ d'_1\ d_2,t')$. Hence, taking $d' = \mathsf{Seq}\ d'_1\ d_2$, we obtain (G1) (by satisfying the $\theta\ c'\ d'$ part of the disjunction), as desired.
- Case 2.2: $s' \sim t$ and $c'_1 \approx_{01\mathsf{T}} d_1$. Hence, taking $d' = \mathsf{Seq}\ d_1\ d_2$, we obtain (G2) (by satisfying the $\theta\ c'\ d$ part of the disjunction), as desired.

Occasionally, we have to strengthen the coinduction hypothesis. E.g., for (3), the relation we prove to be a generalized postfixpoint is not $\theta\ c\ d \equiv \exists c_1\ d_1.\ c = \mathsf{While}\ tst\ c_1\ \wedge$

$d = \mathsf{While}\ tst\ d_1 \wedge c_1 \approx_{\mathsf{s}} d_1$, but rather $\theta\ c\ d\ \vee\ \theta'\ c\ d$, where $\theta'\ c\ d \equiv \exists c_2\ d_2\ c_1\ d_1.\ c = \mathsf{Seq}\ c_2\ (\mathsf{While}\ tst\ c_1)\ \wedge\ d = \mathsf{Seq}\ d_2\ (\mathsf{While}\ tst\ d_1)\ \wedge\ c_2 \approx_{\mathsf{s}} d_2\ \wedge\ c_1 \approx_{\mathsf{s}} d_1$. □

EXAMPLE 4. The informal arguments in Examples 1 and 3 can be made rigorous using the compositionality table in Figure 4 in conjunction with the implication graph in Figure 3. We illustrate the emerging proof methodology for c_4 from Example 1. c_4 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 (row 3 column 5), for $c_4 \approx_{01} c_4$ to hold, 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 (row 3 column 4), since $l := 4$ is compatible. However, the table cannot help (yet) in proving $c_3 \approx_{01} c_3$, because there (in row 4 column 5) 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 attempt 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 (by row 4 column 3) to $\mathsf{discr}\ (\mathsf{Seq}\ (h := 1)\ (h := 2))$ and $\mathsf{discr}\ (h := 3)$, and further (by row 3 column 3) to $\mathsf{discr}\ (h := 1)$, $\mathsf{discr}\ (h := 2)$ and $\mathsf{discr}\ (h := 3)$, all being true thanks to their $\sim$-preservation (by row 2 column 3).

Note that we appeal to the Figure 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 Figure 4. φ is more compositional than ψ w.r.t. Seq.[5] 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 the above is also the lack of compositionality of ψ w.r.t. While (since the semantics of While involves iteration of Seq)—hence the side-condition False in the ψ-While column. False may seem like a very crude approximation, but one cannot hope for anything better without engaging in a static analysis deeper than compositionality. Indeed, as soon as alternative executions of c are allowed to desynchronize w.r.t. termination, their iterations through While may desynchronize completely; an exception is the case when c is in fact discreet, but this is covered in the overall scheme by "falling back" from ψ to discr and applying compositionality of the latter.

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 again 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

[5]Recall from page 7 that interactivity of the considered security notions ensures their Par-compositionality; termination-sensitiveness plays a similar role for Seq-compositionality.

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 so that all executions of the whole pool (including c) from a given state s are 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 of what remains to be executed from c is preserved. Indeed, a violation of (1) or (2) would immediately yield a possibility of nontermination for the whole pool. 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 $\forall i.\ (c_i,s_i)\rightarrow_{\mathsf{C}}(c_{i+1},s_{i+1})$:

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

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

—$c\approx_{\mathsf{T}} d \equiv \mathsf{match}^{\mathsf{C}}_{\mathsf{TMC}}\ (\approx_{\mathsf{T}})\ c\ d\ \wedge\ \mathsf{match}^{\mathsf{T}}_{\mathsf{TMT}}\ c\ d$

And, indeed, $\approx_{\mathsf{T}}$ achieves the targeted properties, as can be shown by an argument similar to those of Propositions 2 and 3:

PROPOSITION 4.
(1) The compositionality facts stated in Figure 4 for φ also hold for $\approx_{\mathsf{T}}$.
(2) $\mathsf{discr}\ c \Longrightarrow c\ \approx_{\mathsf{T}}\ c$ and $\approx_{\mathsf{WT}}\ c \Longrightarrow c\ \approx_{\mathsf{T}}\ c$

PROOF. (1): Similarly to Proposition 3.
(2) Similarly to Proposition 2. □

Note that $\approx_{\mathsf{T}}$ does not require, for the involved programs, termination (a liveness property), but rather preservation of termination (a safety property). In Example 1, c_9 is the program $h' := 0\ ;\ d\ ;\ l := 4$ where d is $\mathsf{while}\ h > 0\ \mathsf{do}\ \{h := h-1\ ;\ h' := h'+1\}$. We can establish that $c_9 \approx_{\mathsf{T}} c_9$:

—Intuitively, two alternative executions E_0 and E_1 of c_9 starting with different initial values of h[6] can be continuously synchronized as required by $\approx_{\mathsf{T}}$'s mathcers (continuation step against multiple continuation steps and termination step against multiple continuation steps followed by termination step) regardless of the intervention of the environment, provided this intervention does not break termination:
  —as long as E_i is still in the while loop, E_{1-i} can also stay in the while loop;
  —as soon as E_i exits the while loop, thanks to preservation of termination, E_{1-i} can also exit the while loop; hence E_{1-i} can synchronize with E_i on the visible step $l := 4$.

[6]We may ignore (from the leakage point of view) the other high variable, h', since it is immediately rewritten.

—Formally, the fact follows by the table-and-graph method: $h' := 0$ and d are both discreet, hence their sequential composition is discreet, a fortiori self bisimilar w.r.t. $\approx_{\mathsf{T}}$; $l := 4$ is self isomorphic, again a fortiori self bisimilar w.r.t. $\approx_{\mathsf{T}}$; by the table, the composition c_9 is self bisimilar w.r.t. $\approx_{\mathsf{T}}$.

In the above "formal" argument, we used that $\approx_{\mathsf{T}}$ is implied by both discreetness and self isomorphism and is compositional w.r.t. sequential composition, which is not the case for the other security notions. In fact, c_9 is not secure according to these other notions—it is not self weakly bisimilar essentially because, depending on the initial value of h, the execution may loop (if $h < 0$) or may terminate assigning 4 to l (if $h \geq 0$).

But neither is $\approx_{\mathsf{T}}$ implied by $\approx_{\mathsf{W}}$, not even by $\approx_{01}$. The program e, given by $\mathsf{if}\ h > 0\ \mathsf{then}\ \{l := 1\ ;\ h := 0\}\ \mathsf{else}\ l := 1$, is self 01-bisimilar. But e is not self bisimilar according to $\approx_{\mathsf{T}}$; this is essentially because, branching on the high variable h, the left branch may take the continuation step $l := 1$ while the right branch only has a termination step $l := 1$ available, thus failing to satisfy $\approx_{\mathsf{T}}$'s termination-versus-termination matcher.

The fact that programs like e are self 01-bisimilar cannot be established by a syntactic type-system-like criterion, as it depends on semantic behavior—to see this, imagine that, instead of $l := 1$, we have $l := E$ on the left branch and $l := F$ on the right branch, where E and F are complex expressions that happen to evaluate to the same value in this particular context. In fact, as we show in the next section, if we only apply syntactic criteria based on compositionality (which is what most security type systems from the literature do), there is nothing we can prove about $\approx_{\mathsf{W}}$ or $\approx_{01}$ that we cannot also prove about $\approx_{\mathsf{T}}$—this is because $\approx_{\mathsf{T}}$ is better located in the table-and-graph scheme.

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.

6.1 Extraction of Syntactic Criteria

Before listing these clauses, we first simplify the Figure 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 Figure 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. Figure 5 shows this new graph, where $\approx_{\mathsf{T}}$ is also integrated. In the Figure 4 table, we also redefine ψ_{T} by redirecting $\approx_{01}$ to siso:

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

Let us introduce some notation for the Figure 4 table and the Figure 5 graph. A (syntactic) *constructor Cns* is any of the following: Seq, If *tst* where *tst* ∈ **test**, While *tst* where *tst* ∈ **test**, Par. In addition, for uniformity, we also introduce a constructor Atm *atm* for every *atm* ∈ **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*).

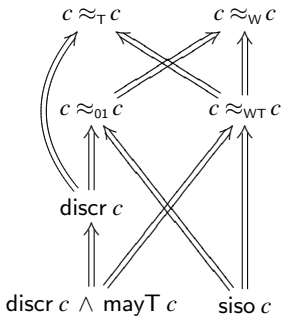

Figure 5: Simplified implication graph of security notions

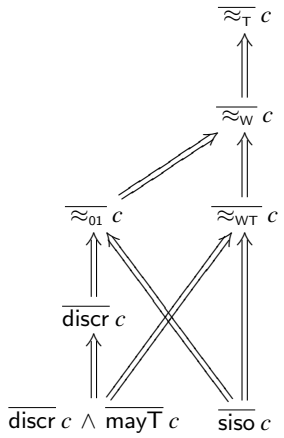

Figure 6: Syntactic implications

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 $\mathsf{Pred}\ \chi$ denote its set of predecessors. E.g., $\mathsf{Pred}\ \mathsf{siso} = \emptyset$, $\mathsf{Pred} \approx_{01} = \{\mathsf{discr}, \mathsf{siso}\}$, $\mathsf{Pred} \approx_{\mathsf{W}} = \{\approx_{01}, \approx_{\mathsf{WT}}\}$.

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

—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:

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

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

We define the above $\overline{\chi}$'s one at a time, traversing the graph in Figure 5 bottom-up, in some order consistent with the graph. Thus, at the time we define some $\overline{\chi}$, we have $\overline{\chi'}$ available for all $\chi' \in \mathsf{Pred}\ \chi$.

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

(1) $\overline{\mathsf{discr}}\ (\mathsf{If}\ \mathit{tst}\ \ c_1\ c_2) = (\overline{\mathsf{discr}}\ c_1 \wedge \overline{\mathsf{discr}}\ c_2)$.

$$(2)\quad \overline{\mathsf{siso}}\ (\mathsf{If}\ tst\ c_1\ c_2) = \begin{cases} \overline{\mathsf{siso}}\ c_1\ \wedge\ \overline{\mathsf{siso}}\ c_2, & \text{if } \mathsf{cpt}\ tst \\ \mathsf{False}, & \text{otherwise.} \end{cases}$$

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

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

(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 $\overline{\approx_{\mathsf{WT}}}$.)

Note that the functions $\overline{\chi}$ are not entirely syntactic, since for test and atoms they employ the semantic notions of compatibility and preservation. For instance, the atom $l := h - h$ is $\sim$-compatible, since it merely assigns 0 to l, but a standard syntactic analysis would reject it since its left-hand side contains h and hence it may depend on h. In what follows, we ignore this aspect and pretend that $\sim$-compatibility and $\sim$-preservation are already "syntactic"; but the results of this paper still hold if one replaces these notions with any syntactic notions that approximate them.

6.2 Soundness

We can show that the automatically extracted syntactic criteria are valid approximations of the semantic notions:

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

PROOF. We prove $\overline{\chi}\ c \Longrightarrow \chi\ c$ separately, for each χ, observing the implication-graph order: we prove the fact for χ only after it has been proved for all $\chi' \in \mathsf{Pred}\ \chi$. With the semantic implication and compositionality results in place, the facts follow by structural induction on c. Schematically, assuming inductively that c has the form $Cns\ c_1 \ldots c_k$ and $\overline{\chi}\ c$ holds, we distinguish 2 cases:

—If $\mathsf{side}_{\chi,Cns}(c_1,\ldots,c_k)$ holds, then $\overline{\chi}\ c = \overline{\mathsf{rcond}_{\chi,Cns}}(c_1,\ldots,c_k)$. By the induction hypothesis $\overline{\mathsf{rcond}_{\chi,Cns}}(c_1,\ldots,c_k)$ implies $\mathsf{rcond}_{\chi,Cns}(c_1,\ldots,c_k)$, which in turn, by Proposition 3 or 4(1), implies $\chi\ (Cns\ c_1 \ldots c_k)$, i.e., $\chi\ c$. Thus, $\overline{\chi}\ c \Longrightarrow \chi\ c$.

—If $\mathsf{side}_{\chi,Cns}(c_1,\ldots,c_k)$ does not hold, then $\overline{\chi}\ c = \bigvee_{\chi' \in \mathsf{Pred}\ \chi}\ \overline{\chi'}\ c$. Since we assume the fact proved for the predecessor of χ, $\bigvee_{\chi' \in \mathsf{Pred}\ \chi}\ \overline{\chi'}\ c$ implies $\bigvee_{\chi' \in \mathsf{Pred}\ \chi}\ \chi'\ c$, which in turn, by Proposition 2 or 4(2), implies $\chi\ c$. Thus, again $\overline{\chi}\ c \Longrightarrow \chi\ c$.

This concludes the proof. □

A remarkable property of the $\overline{\chi}$'s is that they preserve the Figure 5 hierarchy of χ's. Even more remarkable is that they actually refine it. The hierarchy refinement from Figure 5 to Figure 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 $\approx_{01}$.

PROPOSITION 6. The implications listed in Figure 6 hold.

PROOF. Let φ and χ be such that $\overline{\varphi}$ is located below $\overline{\chi}$ in the Figure 6 graph. We have 2 cases.

Case 1: φ is also located below χ in the implication graph of Figure 5, (which means, by Proposition 3 or 4(1), that $\forall d.\ \varphi\ d \Longrightarrow \chi\ d$). There are 2 crucial facts to notice about the recursive clauses for $\overline{\varphi}$ and $\overline{\chi}$:

(1) $\mathsf{side}_{\chi,Cns}(c_1,\ldots,c_k) \Longrightarrow \mathsf{side}_{\varphi,Cns}(c_1,\ldots,c_k)$
(2) Assuming $\mathsf{side}_{\chi,Cns}(c_1,\ldots,c_k)$ holds and $\forall i \in \{1,\ldots,k\}.\ \overline{\varphi}\ c_i \Longrightarrow \overline{\chi}\ c_i$, then $\overline{\mathsf{rcond}_{\varphi,Cns}}(c_1,\ldots,c_k) \Longrightarrow \overline{\mathsf{rcond}_{\chi,Cns}}(c_1,\ldots,c_k)$.

We now prove $\overline{\varphi}\ c \Longrightarrow \overline{\chi}\ c$ by induction on c. Assume inductively that c has the form $Cns\ c_1 \ldots c_k$. If $\mathsf{side}_{\chi,Cns}(c_1,\ldots,c_k)$ does not hold, then $\overline{\chi}\ c = \bigvee_{\chi' \in \mathsf{Pred}\ \chi} \overline{\chi'}\ c$ and the proof is done, since $\varphi \in \mathsf{Pred}\ \chi$. On the other hand, if $\mathsf{side}_{\chi,Cns}(c_1,\ldots,c_k)$ holds, then ,by (1), $\mathsf{side}_{\varphi,Cns}(c_1,\ldots,c_k)$ also holds. Thus $\overline{\varphi}\ c = \overline{\mathsf{rcond}_{\varphi,Cns}}(c_1,\ldots,c_k)$ and $\overline{\chi}\ c = \overline{\mathsf{rcond}_{\chi,Cns}}(c_1,\ldots,c_k)$, and hence the desired fact follows from the induction hypothesis and (2).

Case 2: $\varphi = \approx_{\mathsf{W}}$ and $\chi = \approx_{\mathsf{T}}$. We first prove
(3) $\overline{\approx_{01}}\ c \Longrightarrow \overline{\approx_{\mathsf{T}}}\ c$,
by induction on c. Assume inductively that c has the form $Cns\ c_1 \ldots c_k$.

First assume $\mathsf{side}\ \approx_{01},_{Cns}(c_1,\ldots,c_k)$ holds. Then so does $\mathsf{side}\ \approx_{\mathsf{T}},_{Cns}(c_1,\ldots,c_k)$ (immediate verification). Thus, $\overline{\approx_{01}}\ c = \overline{\mathsf{rcond}\ \approx_{01},_{Cns}}(c_1,\ldots,c_k)$ and $\overline{\approx_{\mathsf{T}}}\ c = \overline{\mathsf{rcond}\ \approx_{\mathsf{T}},_{Cns}}(c_1,\ldots,c_k)$. If $Cns \neq \mathsf{Seq}$, then the desired fact follows by the induction hypothesis. If $Cns = \mathsf{Seq}$, then the desired fact follows from the induction hypothesis together with two facts already proved at Case 1, namely, $\forall d.\ \overline{\mathsf{discr}}\ d \Longrightarrow \overline{\approx_{\mathsf{T}}}\ d$ and $\forall d.\ \overline{\approx_{\mathsf{WT}}}\ d \Longrightarrow \overline{\approx_{\mathsf{T}}}\ d$.

Next assume $\mathsf{side}\ \approx_{01},_{Cns}(c_1,\ldots,c_k)$ does not hold. Then $\overline{\approx_{01}}\ c = \bigvee_{\chi' \in \mathsf{Pred}} \approx_{01}\ \overline{\chi'}\ c$, and from Case 1 we know that $\overline{\chi'}\ c \Longrightarrow \overline{\approx_{\mathsf{T}}}\ c$ for all $\chi' \in \mathsf{Pred}$, hence the desired fact follows.

Finally, we prove $\overline{\approx_{\mathsf{W}}}\ c \Longrightarrow \overline{\approx_{\mathsf{T}}}\ c$ by induction on c. The proof goes similarly to that of (3), but now using $\forall d.\ \overline{\approx_{01}}\ d \Longrightarrow \overline{\approx_{\mathsf{T}}}\ d$ (i.e., fact (3)) and $\forall d.\ \overline{\approx_{\mathsf{WT}}}\ d \Longrightarrow \overline{\approx_{\mathsf{T}}}\ d$ in case $\mathsf{side}\ \approx_{\mathsf{W}},_{Cns}(c_1,\ldots,c_k)$ holds and $Cns = \mathsf{Seq}$. □

Note that the rationale behind why $\overline{\approx_{\mathsf{T}}}$ "wins the battle" against $\overline{\approx_{01}}$ and $\overline{\approx_{\mathsf{W}}}$, advancing above them in the syntactic graph (in spite of its semantic counterpart $\approx_{\mathsf{T}}$ not being above $\approx_{01}$ or $\approx_{\mathsf{W}}$ the semantic graph), is that, on the one hand, $\approx_{\mathsf{T}}$ is more compositional than $\approx_{01}$ and $\approx_{\mathsf{W}}$, and, on the other, it dominates their (semantic) graph predecessors.

6.3 Connections with Syntactic Criteria from the Literature

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.

In what follows, we invoke syntactic criteria from the literature. Usually these results are presented as "security type systems", while we employ a syntax-directed presentation as structurally recursive functions $\overline{\chi}$ on the syntax of programs. For readers familiar with language-based noninterference, it should be intuitively obvious that the two styles of presentation are equivalent—we examine this equivalence in detail in the technical report [28].

$\overline{\mathsf{siso}}\ c$ corresponds to a type system from Smith and Volpano [42] 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 [42]. Here, high tests are allowed at If provided the

branches are discreet. However, high tests are entirely disallowed at While:

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

The above harsh condition on While is the starting point of work by Boudol and Castellani in [6, 7], 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 $\overline{\mathsf{discr}}$:

$$\overline{\approx_{01}}\ (\mathsf{While}\ tst\ c)\ =\ \overline{\mathsf{discr}}\,(\mathsf{While}\ tst\ c) \vee \overline{\mathsf{siso}}\,(\mathsf{While}\ tst\ c)\ =\ \overline{\mathsf{discr}}\,c \vee (\mathsf{cpt}\ tst \wedge \overline{\mathsf{siso}}\,c)$$

(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 [5], 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 [5] 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 theorems for the proposed type systems (results corresponding to Proposition 5) 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. Note that these criteria are expressed not *relationally*, as type systems, but *functionally*, as recursively defined operators on syntax, which makes them trivially syntax-directed.

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 [46], 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_9 and $c_{10} \equiv c_9 \parallel l := 1 \parallel h := 0 \parallel h := h+1$ from Example 1 (as shown by immediate computation with the clauses of $\overline{\approx_{\mathsf{T}}}$ and its predecessors in the graph), 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_9 and c_{10} 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 c_{10} out of fear that its c_9 component may not terminate, which could in principle yield the pipelining of the c_9 termination channel into a standard channel for c_{10}. Termination knowledge excludes such behavior, and this is where the new criterion $\overline{\approx_{\mathsf{T}}}$ is advantageous. In summary, $\overline{\approx_{\mathsf{T}}}$ is superior to the other criteria if and only if the termination behavior can be analyzed in some way.

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 Section 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 Section 5) iff there exist no infinite (c,s)-traces.

We first need to establish a property of the "weak" notions of bisimilarity concerning matching multi-steps by suitable multi-steps:

PROPOSITION 7.

(1) *If* $c \approx_{wT} d$, $s \sim t$ *and* $(c,s) \rightarrow_c^* (c',s')$, *then there exist* d' *and* t' *such that* $(d,t) \rightarrow_c^* (d',t')$, $s' \sim t'$, *and* $c' \approx_{wT} d'$.

(2) *If* $c \approx_{wT} d$, $s \sim t$ *and* $(c,s) \rightarrow_T^* s'$, *then there exists* t' *such that* $(d,t) \rightarrow_T^* t'$ *and* $s' \sim t'$.

(3) *If* $c \approx_w d$, $s \sim t$ *and* $(c,s) \rightarrow_c^* (c',s')$, *then one of the following holds:*
—$\mathsf{discr}\ c'$ *holds and there exists* t' *such that* $(d,t) \rightarrow_T^* t'$ *and* $s' \sim t'$.
—*There exist* d' *and* t' *such that* $(d,t) \rightarrow_c^* (d',t')$, $s' \sim t'$, *and* $c' \approx_w d'$.

(4) *If* $c \approx_w d$, $s \sim t$ *and* $(c,s) \rightarrow_T^* s'$, *then one of the following holds:*

—There exists t' such that $(d,t)\rightarrow_{\tau}^{*}t'$ and $s'\sim t'$.
—There exist d' and t' such that $(d,t)\rightarrow_{c}^{*}(d',t')$, $s'\sim t'$, and $\mathsf{discr}\ d$.

(5) *If* $\mathsf{mustT}\ c$, $\mathsf{mustT}\ d$, $c\approx_{\tau}d$, $s\sim t$ *and* $(c,s)\rightarrow_{c}^{*}(c',s')$, *then there exist d' and t' such that $(d,t)\rightarrow_{c}^{*}(d',t')$, $s'\sim t'$, and $c'\approx_{\tau}d'$.*

(6) *If* $\mathsf{mustT}\ c$, $\mathsf{mustT}\ d$, $c\approx_{\tau}d$, $s\sim t$ *and* $(c,s)\rightarrow_{\tau}^{*}s'$, *then there exists t' such that $(d,t)\rightarrow_{\tau}^{*}t'$ and $s'\sim t'$.*

PROOF. (1), (3), (5): By a straightforward induction on the the definition of $\rightarrow_{c}^{*}$, using the definitions of the matchers involved in $\approx_{\mathsf{WT}}$, $\approx_{\mathsf{W}}$ and $\approx_{\mathsf{T}}$.
(2), (4), (6): From (1), (3) and (5), respectively, using again the definitions of the involved matchers. □

As a technical tool for the proofs that follow, we define a *partial trace* to be a finite sequence of the form $(c_0,s_0),(c_1,s_1),\ldots,(c_{n-1},s_{n-1})$ such that $(c_0,s_0)=(c,s)$ and $(c_i,s_i)\rightarrow_{\mathsf{c}}(c_{i+1},s_{i+1})$ for all $i<n-1$. Note that any finite trace consists of a partial trace as above together with a terminating state s_n, such $(c_{n-1},s_{n-1})\rightarrow_{\mathsf{T}}s_n$.

We can now to prove the following about the termination-sensitive bisimilarities:

PROPOSITION 8.
(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 $c\approx_{\mathsf{01T}}c$ 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 $\mathsf{fstate}(tr')\sim\mathsf{fstate}(tr)$.

PROOF. As usual, we prove the facts in the more general binary format, e.g.:
If $c\approx_{\mathsf{S}}d$ and $s\sim t$, then, for every finite (c,s)-trace tr, there exists a finite (d,t)-trace tr' with $\mathsf{fstate}(tr')\sim\mathsf{fstate}(tr)$ and $\mathsf{length}(tr')=\mathsf{length}(tr)$.
(1) and (2): We first prove these facts for partial traces instead of finite traces, by a straightforward induction on the partial trace tr. Then the facts for finite traces follow from $\approx_{\mathsf{S}}$ and $\approx_{\mathsf{01T}}$. matching terminating steps with terminating steps.
(3): From Proposition 7(2), using the correspondence between terminating traces starting at (c,s) and ending at s' and terminating multi-steps $(c,s)\rightarrow_{\mathsf{T}}^{*}s'$. □

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:

PROPOSITION 9. If $\forall s'.\ \mathsf{mustT}\ (c,s')$ holds, then Proposition 8(3) holds with any of $\approx_{01}$ and $\approx_{\mathsf{W}}$ substituted for $\approx_{\mathsf{WT}}$.

PROOF. Again, we prove the more general, binary facts, e.g.: If $\forall t'.\ \mathsf{mustT}\ (d,t')$ holds, $c\approx_{\mathsf{W}}d$ and $s\sim t$, then, for every finite (c,s)-trace tr, there exists a finite (d,t)-trace tr' with $\mathsf{fstate}(tr')\sim\mathsf{fstate}(tr)$.

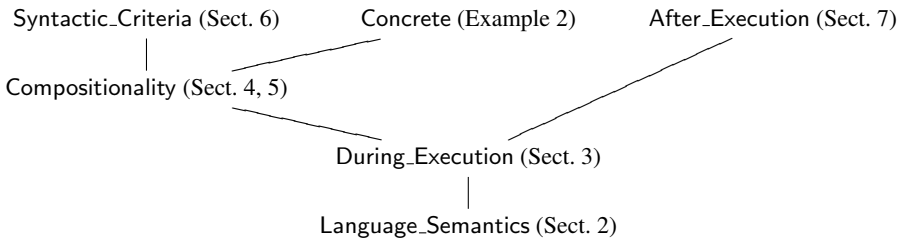

Fig. 7: Main theory structure of the formalization

Assuming $c \approx_{\mathsf{w}} d$, $s \sim t$ and given a trace tr starting at (c,s) and ending at s', by Proposition 7(4) we have two cases:

—There is a trace starting at (d,t) and ending at some t' with $s' \sim t'$. Then we are done.

—There is a partial trace tr' starting at (d,t) and ending at some (d',t') with $s' \sim t'$ and $\mathsf{discr}\ d'$. By our must-terminate assumption and Proposition 1, there is a trace tr'' from (d',t') to some state t''. By the discreetness of d', we have $t'' \sim t'$, hence $t'' \sim s'$. Therefore, $tr' \cdot tr''$, the concatenation of tr' and tr'', is the desired trace matching tr.

For $\approx_{01}$, the proof is similar, but the first step, namely, that obtaining the partial trace tr', requires induction on tr. □

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 Section 5, it is weaker than that of $\approx_{\mathsf{WT}}$ (Proposition 8(3)), but stronger than that of $\approx_{01}$ and $\approx_{\mathsf{w}}$ (Proposition 9):

PROPOSITION 10. If $\mathsf{mustT}\ (c,s)$ holds, then Proposition 8(3) holds with $\approx_{\mathsf{T}}$ substituted for $\approx_{\mathsf{WT}}$.

PROOF. We prove the binary fact: If $\mathsf{mustT}\ (c,s)$, $\mathsf{mustT}\ (d,t)$, $c \approx_{\mathsf{w}} d$ and $s \sim t$, then, for every finite (c,s)-trace tr, there exists a finite (d,t)-trace tr' with $\mathsf{fstate}(tr') \sim \mathsf{fstate}(tr)$. The fact follows from Proposition 7(6). □

8. ISABELLE FORMALIZATION

The results presented in this paper have been formalized in Isabelle/HOL—the development is publicly available in the Isabelle Archive of Formal Proofs [29].

The Isabelle proofs are essentially detailing the proof sketches shown in the paper. What we deem "trivial" or "straightforward" in the proof sketches is usually handled automatically by Isabelle's "auto" tactic.

The formal proofs by coinduction, especially those of compositionality facts, were very tedious, having to expand the definition of matchers, apply inversion rules and consider several cases. There were three choices that helped keep the proof sizes under control to some extent:

—Taking the greatest fixpoints in the space of *symmetric* binary relations (rather than in that of arbitrary binary relations) allowed us to only consider half of the bisimilarity game: a right step simulating a left step, but not vice versa. In exchange, we had to

prove that the bisimulation candidates (i.e., the relations to be proved postfixpoints) were symmetric, which was always immediate.

—Working with generalized postfixpoints (rather than plain postfixpoints) allowed us to employ smaller bisimulation candidates, namely, ones that "stop" when reaching bisimilar items. This helped reduce the size of the proofs for operators such as sequential composition and while.

—Using the Sledgehammer tool for deploying external automatic theorem provers [26] helped automate some of the tedious $\forall\exists$ reasoning specific to coinduction/bisimilarity proofs. This was especially true for the simpler security predicates siso and discr.

Proofs by coinduction are still not supported in Isabelle as well as they could be. Many goals had a special form, namely, conditional equations with equality being bisimilarity. To put such goals in a form suitable for coinduction, we had to rephrase them employing existential quantification, as in the proof sketch of Proposition 3; the existential quantification is later eliminated. A more intuitive proof would proceed directly to unfolding the terms of the equation according to the operational semantics—an Isabelle coinduction tactic providing direct unfolding for goals of this pattern would allow shorter and cleaner proofs. This technology is under development within Isabelle's (co)datatype project [44].

The compositionality results account for about half of the size of the overall development: 2729 lines out of a total of 6221. Of course, not only the length of the proofs, but also the sheer number of facts stated in Figure 4's table is a contributing factor to this size.

On the bright side, the proofs of the soundness of, and implications between, the syntactic criteria go very smoothly. For instance, here is the whole formal proof of Proposition 6:

```
theorem SC_siso_imp_SC_WbisT[intro]: "SC_siso c ⟹ SC_WbisT c"
  by (induct c) auto

theorem SC_discr_imp_SC_WbisT[intro]:
"SC_mayT c ⟹ SC_discr c ⟹ SC_WbisT c"
  by (induct c) (auto simp: presAtm_compatAtm)

theorem SC_discr_imp_SC_ZObis[intro]: "SC_discr c ⟹ SC_ZObis c"
  by (induct c) (auto simp: presAtm_compatAtm)

theorem SC_siso_imp_SC_ZObis[intro]: "SC_siso c ⟹  SC_ZObis c"
  by (induct c) auto

theorem SC_ZObis_imp_SC_Wbis[intro]: "SC_ZObis c ⟹ SC_Wbis c"
  by (induct c) auto

theorem SC_WbisT_imp_SC_Wbis[intro]: "SC_WbisT c ⟹  SC_Wbis c"
  by (induct c) auto

theorem SC_discr_imp_SC_BisT[intro]: "SC_discr c ⟹  SC_BisT c"
  by (induct c) (auto simp: presAtm_compatAtm)

theorem SC_WbisT_imp_SC_BisT[intro]: "SC_WbisT c ⟹  SC_BisT c"
  by (induct c) auto
```

```
theorem SC_ZObis_imp_SC_BisT[intro]: "SC_ZObis c ⟹  SC_BisT c"
  by (induct c) auto

theorem SC_Wbis_imp_SC_BisT[intro]: "SC_Wbis c ⟹ SC_BisT c"
  by (induct c) (auto split: split_if_asm)
```

Above, the syntactic analogues of the semantics notions, indicated in the paper by overlining, e.g., $\overline{\text{discr}}$, are prefixed by "SC" (from "syntactic criterion"), e.g., SC_discr, SC_WbisT. The proofs go fully automatically after issuing a structural induction on the command c. However, as observed in the informal proof, the order of stating these facts is important: one needs to start at the bottom of Figure 6's graph and advance upwards. In the formalization, a proved fact is declared as an introduction rule, so that proofs of facts located above in the graph can use them automatically (via the "auto" method).

An interesting question is whether we would have been better off had we worked schematically, as we informally did in the presentation of this paper: stating and proving facts universally quantified over the set of the considered bisimilarities, having an Isabelle operator $_{-T}$ that maps each termination-insensitive notion ψ to its termination-sensitive counterpart ψ_T, etc. In other words, would it have helped to employ a deep embedding of our informal schemas? We believe the answer is no, due to the substantial bureaucracy associated with switching from such a deep embedding back to the shallow(er) end results.

9. 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 $\approx_{\mathsf{T}}$.

Our approach has important precursors in the literature. [38] 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, [38] does not pursue this idea systematically or devise a general technique as we do in this paper.

Another line of work focusing on compositionality in language-based security is Morgan's shadow semantics [23, 24], which defines a Hoare-style program logic together with a notion of refinement coping with security. This work, inspired by knowledge logic [12], goes beyond type systems and pure noninterference, allowing one to express and prove interactively complex declassification properties. The approach is currently restricted to nondeterministic sequential programs.

Our bisimilarity-based treatment also employs insight from process algebra [22] in general and from process algebra approaches to noninterference [11] in particular. In system-based security, [12, 15, 21] provide general frameworks for trace-based system security, the last two having a special focus on compositionality and the first also incorporating probabilistic systems.

While the execution traces of programs form systems in the sense of these frameworks, language-based security has developed a specific set of concepts and methods (some of them reviewed in this paper) whose relationship with system security is a nontrivial problem, analyzed in [16, 17, 45]. The Goguen-Meseguer-style trace-based noninterference,

an adaptation of which we also employ in this paper, has been challenged in [10], which proposes a refinement of the notion using partial orderings of events to account for collaborating adversaries.

We only gave small examples of programs used to distinguish between the various notions of security. However, there is no fundamental difficulty with applying these results to much larger programs—our network of syntactic criteria from Section 6 employ a fairly efficient traversal of the program syntax. More substantial difficulties with extending the analysis to realistic programs are the programming-language and programming-environment features. Themes missing from our compositionality framework are probabilistic noninterference [19, 39–41], dynamic thread creation [19, 38, 47] and scheduler independence [7, 19, 38, 47], known to be particularly problematic w.r.t. noninterference. We have dedicated them a separate study in [30], where we identify a class of schedulers that guarantee probabilistic noninterference of a thread pool assumed to satisfy possibilistic noninterference; the guarantee is valid in the presence of dynamic thread creation. No particular programming language is considered there, but the abstract relationship between scheduler and thread pool is studied in depth.

Amongst previous formalizations of noninterference-like properties, the most closely related to our work are Snelting and Wasserrab's formalization of the original Volpano-Smith-Irvine type system for single-threaded programs [43] and Barthe and Nieto's formalization of the Boudol-Castellani approach [4], both in Isabelle. Recently, David Cock [8] has formalized in Isabelle noninterference of a probabilistic scheduler and has discussed its integration with the seL4 operating system kernel verification [14]. In the Coq proof assistant, Nanevski, Banerjee and Garg have formalized the Relational Hoare Type Theory [25], a shallowly embedded framework for establishing security properties on top of a realistic sequential language featuring mutable state and dynamic memory allocation.

An 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 [3, 9, 18, 23, 24]. This would follow a rely-guarantee paradigm [13], with information about the environment made available to individual threads by suitably relaxing interactivity. A step towards this direction is made by our bisimilarity $\approx_{\mathsf{T}}$, where such context information is termination, but could in principle be any liveness property.

Acknowledgements. Jasmin Blanchette made lots of suggestions that have significantly improved the presentation of the paper. Benedict Nordhoff and Peter Lammich noticed various technical typos. The anonymous reviewers of both the conference version and this journal version suggested many improvements.

References

[1] J. Agat. Transforming out timing leaks. In *POPL*, pages 40–53, 2000.

[2] J. Baeten and W. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[3] 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.

[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. In *ICALP*, pages 382–395, 2001.

[7] G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1-2):109–130, 2002.

[8] D. Cock. Practical probability: Applying pgcl to lattice scheduling. To appear in ITP 2013.

[9] Á. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In *SPC*, pages 193–209, 2005.

[10] K. Engelhardt. A note on noninterference in the presence of colluding adversaries (preliminary report). ARSPA-WITS'10. `http://www.cse.unsw.edu.au/~kaie/Publications/Papers/E2010ARSPAWITS.pdf`.

[11] R. Focardi and R. Gorrieri. Classification of security properties (part i: Information flow). In *FOSAD*, pages 331–396, 2000.

[12] J. Y. Halpern and K. R. O'Neill. Secrecy in multiagent systems. *ACM Trans. Inf. Syst. Secur.*, 12(1), 2008.

[13] C. B. Jones. Specification and design of (parallel) programs. In *IFIP Congress'83*, pages 321–332, 1983.

[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] H. Mantel. On the composition of secure systems. In *IEEE Symposium on Security and Privacy*, pages 88–, 2002.

[16] H. Mantel and A. Sabelfeld. A generic approach to the security of multi-threaded programs. In *CSFW*, pages 126–, 2001.

[17] H. Mantel and A. Sabelfeld. A unifying approach to the security of distributed and multi-threaded programs. *Journal of Computer Security*, 11(4):615–676, 2003.

[18] H. Mantel, D. Sands, and H. Sudbrock. Assumptions and guarantees for compositional noninterference. In *CSF'11*, pages 218–232, Cernay-la-Ville, France, 2011.

[19] H. Mantel and H. Sudbrock. Flexible scheduler-independent security. In *ESORICS*, pages 116–133, 2010.

[20] 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.

[21] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. pages 79–93, May 1994.

[22] R. Milner. *Communication and concurrency*. Prentice Hall, 1989.

[23] C. Morgan. The shadow knows: Refinement and security in sequential programs. *Sci. Comput. Program.*, 74(8):629–653, 2009.

[24] C. Morgan. Compositional noninterference from first principles. *Formal Asp. Comput.*, 24(1):3–26, 2012.

[25] A. Nanevski, A. Banerjee, and D. Garg. Verification of information flow and access control policies with dependent types. In *Symposium on Security and Privacy*, pages 165–179, 2011.

[26] 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.

[27] G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

[28] A. Popescu. Security type systems as recursive predicates. Technical report. Available at `http://arxiv.org/abs/1308.3472`.

[29] A. Popescu and J. Hölzl. Possibilistic noninterference. *Archive of Formal Proofs*, 2012. `http://afp.sourceforge.net/entries/Possibilistic_Noninterference.shtml`.

[30] A. Popescu, J. Hölzl, and T. Nipkow. Noninterfering schedulers–when possibilistic noninterference implies probabilistic noninterference. To appear in CALCO 2013.

[31] A. Popescu, J. Hölzl, and T. Nipkow. Proving concurrent noninterference. In *CPP*, pages 109–125, 2012.

[32] 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.

[33] 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.

[34] 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.

[35] A. Sabelfeld. Confidentiality for multithreaded programs via bisimulation. In *International Conference on Perspectives of System Informatics*, LNCS, pages 260–273, 2003.

[36] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[37] 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.

[38] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *IEEE Computer Security Foundations Workshop*, pages 200–214, 2000.

[39] G. Smith. A new type system for secure information flow. In *IEEE Computer Security Foundations Workshop*, pages 115–125, 2001.

[40] G. Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *IEEE Computer Security Foundations Workshop*, pages 3–13, 2003.

[41] G. Smith. Improved typings for probabilistic noninterference in a multi-threaded language. *Journal of Computer Security*, 14(6):591–623, 2006.

[42] 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.

[43] G. Snelting and D. Wasserrab. A correctness proof for the Volpano/Smith security typing system. *Archive of Formal Proofs*, 2008, 2008.

[44] D. Traytel, A. Popescu, and J. C. Blanchette. Foundational, compositional (co)datatypes for higher-order logic—Category theory applied to theorem proving. In *LICS 2012*, pages 596–605. IEEE, 2012.

[45] R. van der Meyden and C. Zhang. A comparison of semantic models for noninterference. *Theor. Comput. Sci.*, 411(47):4123–4147, 2010.

[46] 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.

[47] S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *IEEE Computer Security Foundations Workshop*, pages 29–43, 2003.