Verifying Security Policies in Multi-agent Workflows with Loops

Bernd Finkbeiner
finkbeiner@cs.uni-saarland.de
CISPA, Saarland University

Christian Müller
christian.mueller@in.tum.de
Technische Universität München

Helmut Seidl
seidl@in.tum.de
Technische Universität München

Eugen Zălinescu
eugen.zalinescu@in.tum.de
Technische Universität München

ABSTRACT

We consider the automatic verification of information flow security policies of web-based workflows, such as conference submission systems like EasyChair. Our workflow description language allows for loops, non-deterministic choice, and an unbounded number of participating agents. The information flow policies are specified in a temporal logic for hyperproperties. We show that the verification problem can be reduced to the satisfiability of a formula of first-order linear-time temporal logic, and provide decidability results for relevant classes of workflows and specifications. We report on experimental results obtained with an implementation of our approach on a series of benchmarks.

1 INTRODUCTION

Web-based workflow systems often have critical information flow policies. For example, in a conference management system like EasyChair, the information about a certain paper must be kept secret from all program committee (PC) members who have declared a conflict of interest for the paper until the acceptance notifications are released by the PC chair.

Verification techniques for workflows (cf. [4, 16, 21]) typically build on classic notions of secrecy such as non-interference [17]. The particular challenge with verifying web-based workflow systems is that here is no fixed set of agents participating in the workflow. Clearly, we would not like to reason about the correctness of a conference management system for every concrete installation for a particular conference, a particular program committee and a particular set of submissions and reports. Instead, we would like to prove a given system once for all — for any possible instantiation and any number of PC members, submitted papers and reports.

We present such a verification approach based on the temporal logic HyperLTL [9]. HyperLTL is a general specification language for temporal hyperproperties, which include common information flow policies like non-interference, and time- and data-dependent declassification. HyperLTL can also express assumptions on the behavior of the agents such as *causality*, i.e. that an agent can only reveal information that was received by the agent at a *previous* point in time. This is important in order to analyze chains of information flows, where a piece of information is transmitted via two or more communications, i.e. where agent A learns about a secret known to agent B, even though B never communicates with A directly; instead, B talks to a third agent C, and, subsequently, C talks to A.

HyperLTL-based workflow verification has been considered before, but only for the restricted case of *loop-free* workflows [16]. Such workflows consist of a fixed finite sequence of steps. Although an arbitrary number of agents may participate in each step, the workflow thus only allows a fixed number of interactions between the agents. This is not realistic: to accurately model, for example, the repeated commenting on papers and reviews during the discussion phase of a conference management system, one needs a loop in the workflow.

We present an automatic verification technique for workflows *with loops*. The general outline of our approach is as follows: We specify the operational semantics of the workflow language in many-sorted first-order linear-time temporal logic (FOLTL). The desired information-flow policy and the assumptions on the agents are expressed in first-order HyperLTL (HyperFOLTL). Combining the two specifications, the existence of a violation of the policy reduces to the satisfiability of a HyperFOLTL formula.

We identify an expressive fragment of many-sorted HyperFOLTL, for which satisfiability is decidable. The fragment subsumes the previously known decidable fragments of FOLTL [22] and of HyperLTL [15]. It also generalizes the Bernays-Schönfinkel fragment of first-order logic [1]. Of particular practical value is that our logic is *many-sorted*, i.e. we distinguish different groups of agents such as authors and program committee members. This allows us to place different assumptions on different groups; it also improves the performance of our decision procedure, because the different sorts are kept separate.

We identify a natural class of workflows, which we call *non-omitting* workflows, where the encoding of the verification problem is in the decidable fragment of HyperFOLTL. We thus obtain a decision procedure for non-omitting workflows. This decidability result in fact turns out to be optimal in the sense that for workflows outside the class, non-interference becomes undecidable. The decidable fragment is also sufficiently expressive to specify common information-flow policies like non-interference. In terms of agent assumptions, we show that the fragment is sufficiently expressive to handle strong assumptions like *stubbornness*, meaning that an agent does not reveal any information, for arbitrary sets of agents, and weaker assumptions, like *causality*, for a fixed finite set of agents. This means that we can decide whether a given number of agents

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '17, October 30-November 3, 2017, Dallas, TX, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134080

can *conspire* to cause a leak, assuming that all other agents do not reveal any information. Again, our decidability result is optimal in the sense that the verification problem for unbounded sets of causal agents turns out to be undecidable: it is impossible to decide whether an unbounded number of agents can conspire to reveal a secret.

We report on experimental results based on an implementation of our approach in the tool NIWO. For example, NIWO has found an attack on a simple conference management system, where two program committee members conspire to leak a secret.

2 PRELIMINARIES

Given a sequence $\bar{\sigma}$, we let σ_n denote its n-th element, and $\bar{\sigma}[n, \infty]$ denote its subsequence from n to ∞, i.e. $\bar{\sigma}[n, \infty] := \sigma_n \sigma_{n+1} \ldots$, assuming $\bar{\sigma}$ is infinite. We sometimes abuse notation and use set notation over sequences. For instance, $|\bar{\sigma}|$ denotes the length of $\bar{\sigma}$.

2.1 First-Order LTL (FOLTL)

A *signature* $\Sigma = (S, C, \mathcal{R}, ar)$ consists of a non-empty and finite set of sorts, finite and disjoint sets C and $\mathcal{R}$ of constant and relation (or predicate) symbols, and arity function $ar : C \cup \mathcal{R} \to S^*$, with $|ar(c)| = 1$ for any $c \in C$, where S^* denotes the set of finite sequences of sorts. For each sort s, we let $\mathcal{V}_s$ be a countably infinite set of variables. We let $\mathcal{V} := \bigcup_{s \in S} \mathcal{V}_s$.

FOLTL *formulas* over the signature $\Sigma = (S, C, \mathcal{R}, ar)$ are given by the grammar

$$\varphi ::= t = t' \mid R(t_1, \ldots, t_k) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x{:}\, s.\, \varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi$$

where t, t', and the t_is range over $\mathcal{V} \cup C$, R ranges over $\mathcal{R}$, s ranges over S, and x ranges over $\mathcal{V}_s$. The symbols X and U denote the usual Next and Until LTL operators. As syntactic sugar, we use standard Boolean connectives such as $\wedge, \rightarrow, \leftrightarrow$, the universal quantifier $\forall x$, and the derived temporal operators F (Eventually) with $\mathsf{F}\varphi := \mathit{true}\,\mathsf{U}\,\varphi$, G (Globally) with $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$, W (Weak Until) with $\varphi \,\mathsf{W}\, \psi := (\varphi \,\mathsf{U}\, \psi) \vee \mathsf{G}\,\varphi$, and R (Release) with $\varphi \,\mathsf{R}\, \psi := \neg(\neg\varphi \,\mathsf{U}\, \neg\psi)$, where $\mathit{true} := (c = c)$ for some $c \in C$.

We only consider well-sorted formulas. We omit their definition, which is as expected; for instance, equality is only allowed over terms of the same sort. We may drop the sort in $\forall x{:}\, s.\, \varphi$ when it is irrelevant or clear from the context and simply write $\forall x.\, \varphi$.

We will sometimes consider that formulas are in *negation normal form*, which is obtained by pushing negation inside until it appears only in front of atomic formulas. When considering this form, the operators $\wedge$, $\forall$, and R are seen as primitives, instead of derived ones. A formula is in *prenex normal form* if it is written as a sequence of quantifiers followed by a quantifier-free part.

To omit parentheses, we assume that Boolean connectives bind stronger than temporal connectives, and unary connectives bind stronger than binary ones, except for the quantifiers, which bind weaker than Boolean ones and stronger than temporal ones.

The set of *free variables* of a formula φ, that is, those that are not in the scope of some quantifier in φ, is denoted by $fv(\varphi)$. A formula without free variables is called *closed* or *ground*. For a term $t \in \mathcal{V} \cup C$, we let $fv(t) := \{t\}$ if $t \in \mathcal{V}$ and $fv(t) := \emptyset$ otherwise.

A *structure* $\mathcal{S}$ over the signature $\Sigma = (S, C, \mathcal{R}, ar)$ consists of a S-indexed family of (finite or infinite) *universes* $U_s \neq \emptyset$ and *interpretations* $R^{\mathcal{S}} \in U_{s_1} \times \ldots U_{s_k}$, for each $R \in C \cup \mathcal{R}$ of sort $(s_1, \ldots, s_k)$. We let $U := \bigcup_{s \in S} U_s$. A *temporal structure* over Σ is a sequence $\bar{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \ldots)$ of structures over Σ such that all structures $\mathcal{S}_i$, with $i \geq 0$, have the same universe family, denoted $(U_s)_{s \in S}$, and rigid constant interpretations, i.e. $c^{\mathcal{S}_i} = c^{\mathcal{S}_0}$, for all $c \in C$ and $i > 0$.

Given a structure, a *valuation* is a mapping $\nu : \mathcal{V} \to U$ with x and $\nu(x)$ of the same sort for any $x \in \mathcal{V}$. For a valuation ν and tuples $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{d} = (d_1, \ldots, d_n)$, where $x_i \in \mathcal{V}_s$ and $d_i \in U_s$ for some sort s, for each i, we write $\nu[\bar{x} \mapsto \bar{d}]$ for the valuation that maps each x_i to d_i and leaves the other variables' valuation unaltered. By $\nu(\bar{x})$ we denote the tuple $(\nu(x_1), \ldots, \nu(x_n))$. We extend this notation by applying a valuation ν also to constant symbols $c \in C$, with $\nu(c) = c^{\mathcal{S}}$.

Let $\bar{\mathcal{S}}$ be a temporal structure over the signature Σ, with $\bar{\mathcal{S}} = (\mathcal{S}_0, \mathcal{S}_1, \ldots)$, φ a formula over Σ, and ν a valuation. We define the relation $\bar{\mathcal{S}}, \nu \models \varphi$ inductively as follows:

$$\begin{array}{lll}
\bar{\mathcal{S}}, \nu \models t = t' & \text{iff} & \nu(t) = \nu(t') \\
\bar{\mathcal{S}}, \nu \models R(\bar{t}) & \text{iff} & \nu(\bar{t}) \in R^{\mathcal{S}_0} \\
\bar{\mathcal{S}}, \nu \models \neg\psi & \text{iff} & \bar{\mathcal{S}}, \nu \not\models \psi \\
\bar{\mathcal{S}}, \nu \models \psi \vee \psi' & \text{iff} & \bar{\mathcal{S}}, \nu \models \psi \text{ or } \bar{\mathcal{S}}, \nu \models \psi' \\
\bar{\mathcal{S}}, \nu \models \exists x.\, \psi & \text{iff} & \bar{\mathcal{S}}, \nu[x \mapsto d] \models \psi, \text{ for some } d \in U \\
\bar{\mathcal{S}}, \nu \models \mathsf{X}\psi & \text{iff} & \bar{\mathcal{S}}[1, \infty], \nu \models \psi \\
\bar{\mathcal{S}}, \nu \models \psi \,\mathsf{U}\, \psi' & \text{iff} & \text{for some } j \geq 0,\ \bar{\mathcal{S}}[j, \infty], \nu \models \psi', \text{and} \\
 & & \quad \bar{\mathcal{S}}[k, \infty], \nu \models \psi, \text{ for all } k \text{ with } 0 \leq k < j
\end{array}$$

A FOLTL formula φ is said to be *satisfiable* iff there exists a temporal structure $\bar{\mathcal{S}}$ and a valuation ν s.t. $\bar{\mathcal{S}}, \nu \models \varphi$. It is said to be *finitely satisfiable* iff there exists a temporal structure $\bar{\mathcal{S}}$ over a finite universe U and a valuation ν s.t. $\bar{\mathcal{S}}, \nu \models \varphi$.

We note that unsorted FOLTL can be seen as sorted FOLTL with just one sort.

2.2 FOLTL Decidability

Since FOLTL subsumes First-Order Logic (FOL), FOLTL is also undecidable. In this paper, we consider formulas of a Bernays-Schönfinkel-like fragment[1] of FOLTL, which we name $\exists^*$FOLTL. To define this fragment we will consider the projection of a sorted FOLTL formula on a sort s, defined as the FOLTL formula obtained by removing all quantifications and terms of a sort different from s. We refer to [1, Definition 17] for the formal definition of the projection, and here we only illustrate it with an example. Given the formula $\exists x{:}\, A.\, \forall y{:}\, B.\, \exists z{:}\, A.\, \neg(x = z) \wedge P(x, y) \wedge \mathsf{G}\, Q(y, z)$, its projection on the sorts A and B are the formulas $\exists x.\, \exists z.\, \neg(x = z) \wedge P_2(x) \wedge \mathsf{G}\, Q_1(z)$ and $\forall y.\, P_1(y) \wedge \mathsf{G}\, Q_2(y)$, respectively.

The $\exists^*$FOLTL fragment of sorted FOLTL consists of those closed formulas φ in negation normal form such that, for each sort s, the projection of φ on s is a formula of the form

$$\exists x_1, \ldots, x_k.\, \varphi'_s$$

[1] The Bernays-Schönfinkel-Ramsey fragment, also called "effectively propositional", is one of the first identified decidable fragments of FOL [8]. It consists of those FOL formulas in prenex normal form having the $\exists^*\forall^*$ quantifier prefix.

with $k \geq 0$ and φ'_s a FOLTL formula containing no existential quantifiers. This definition extends the definition of the Bernays-Schönfinkel-like fragments in [1, 27] from FOL to FOLTL,[2] and of the Bernays-Schönfinkel-like fragment in [22] from unsorted FOLTL to sorted FOLTL. Note that the previous sample formula is in $\exists^*$FOLTL.

We can try to put an arbitrary FOLTL formula in the mentioned form by using the standard transformations that put a FOL formula into prenex normal form, as well as the following equivalences to move existential quantifiers outside of temporal operators:

$$\varphi \,\mathsf{U}\, \exists x.\psi \equiv \exists x.\,(\varphi \,\mathsf{U}\, \psi) \quad \text{and} \quad (\exists x.\psi)\,\mathsf{R}\,\varphi \equiv \exists x.\,(\psi \,\mathsf{R}\, \varphi),$$

assuming that x does not occur free in φ. Note that in particular we have that $\mathsf{F}\,\exists x.\varphi \equiv \exists x.\,\mathsf{F}\,\varphi$. However, the previous equivalences cannot be generalized. For instance, existential quantifiers cannot, in general, be moved over the G operator. Intuitively, $\mathsf{G}\,\exists x.\varphi$ means that "for all time points t, there exists an x such that φ holds at t". Thus, in contrast to FOL, not all FOLTL formulas can be put in prenex normal form.

Theorem 2.1 ($\exists^*$FOLTL Decidability).

(1) *Checking satisfiability of a formula in* $\exists^*$FOLTL *is equivalent to checking* finite *satisfiability of the same formula.*
(2) $\exists^*$FOLTL *is decidable.*

Proof. This proof follows the reasoning for decidability of the Bernays-Schönfinkel-Ramsey fragment of FOL, see e.g. [8].

Consider a closed formula φ in $\exists^*$FOLTL. The formula φ has the form $Q_1 x_1 : s_1 \ldots Q_k x_k : s_k.\,\psi$, where $k \geq 0$, $Q_1, \ldots, Q_k$ is a sequence of quantifiers, and ψ is an FOLTL formula in negation normal form containing no existential quantifiers. We group the sequence $Q_1, \ldots, Q_k$ of quantifiers into maximal subsequences of the form $\exists^*\forall^*$. We let n be the number of such subsequences, and let ψ_i be obtained from φ be removing the first i groups of quantifiers, for $0 \leq i \leq n$. Note that $\varphi_0 = \varphi$ and $\varphi_n = \psi$.

We iteratively transform the formula φ_0 into the formulas ψ_1 to ψ_n. We also build the sets D_s^i of constant symbols of sort s, for each sort s and each i with $1 \leq i \leq n$. Consider step i, with $1 \leq i \leq n$. For each sort s, we pick a set C_s^i of constant symbols whose cardinality is given by the number of existential quantifiers over the sort s in the i-th subsequence, and such that $C_s^i \cap C_s^j = \emptyset$ for any $0 < j < i$. We let $D_s^i = C_s^i \cup D_s^{i-1}$, where D_s^0 is a singleton containing some constant of sort s. For each sort s, and each variable x of sort s bound by an existential quantifier from the i-th group, we remove the existential quantifier and we instantiate x in ψ_i by a corresponding constant from C_s^i. In this way, all existentially quantified variables in ψ_i are instantiated. Next, starting from the top-most universal quantifier in ψ_i, we iteratively replace every subformula of the form $\forall y : s.\,\alpha$ by the finite conjunction over elements of D_s^i, namely, $\bigwedge_{d \in D_s^i} \alpha[y \mapsto d]$. Let ψ_{i+1} be the formula obtained in this manner. Finally, we replace all subformulas of the form $\forall y : s.\,\alpha$ in ψ_n (recall that ψ may have universal quantifiers) by the finite conjunction over elements of D_s^n, as above. Let ψ' be the formula obtained in this manner. It is easy to see that φ is satisfiable iff ψ' is satisfiable. Furthermore, it is also clear that ψ' is satisfiable iff it is finitely satisfiable, as we can pick $U_s = D_s^n$ as the (Herbrand) universes. Note that by construction the universe U_s is non-empty even when there is no existential quantifier over the sort s; in this case $C_s^i = \emptyset$, for all i with $1 \leq i \leq n$. This ends the proof of the first statement of the theorem.

For the second statement of the theorem, note that ψ' contains neither quantifiers, nor variables, and thus its atoms are ground. We transform ψ' into an LTL formula by taking the disjunction over all combinations of equivalence relations over D_s^n (for each sort s) of the formulas obtained by replacing each predicate $R(d_1, \ldots, d_\ell)$ in ψ' with the atomic propositions $R_{(d'_1, \ldots, d'_\ell)}$, where d'_i is the representative of the equivalence class to which d_i belongs. Furthermore, equalities $a = b$ are replaced by *true* if a and b are in the same equivalence class and by *false* otherwise. Clearly, the thus obtained formula is equi-satisfiable with ψ'. We can now conclude by noting that LTL satisfiability is decidable, see e.g. [29]. □

We also note that there are very few decidability results concerning FOLTL. Besides the $\exists^*$FOLTL fragment, the only other decidable fragment we are aware of is the monodic fragment [18], which requires that temporal subformulas have at most one free variable. As will become clear in the next section, this restriction is too strong for our purposes: we cannot encode workflows by FOLTL formulas in this fragment.

3 WORKFLOWS

In this section, we will define a language of *workflows*. Our definition of workflows extends the definition of workflows in [16] with *loops* and *nondeterministic choice*.

Workflows are used to model the interactions of multiple agents with a system. Agent interactions are recorded by relations. Updates of these relations are organized into *blocks*. These describe operations that a subset of the agents can choose to execute to change the relation contents. The most basic construct in the description of workflows is a parameterized guarded update operation to some relation. Such an update is meant to simultaneously be executed for all tuples satisfying the given guard. Some of these updates may also be *optional*, i.e. may also be omitted for some of the tuples that satisfy the guard.

Example 3.1. The workflow in Fig. 1 models the paper reviewing and review updating of EasyChair. In this workflow, all PC members are agents. In a first step, they can declare that they have a conflict of interest with some of the papers. Then, papers are assigned to reviewers as long as they have not declared a conflict with the respective papers. Reviewers are then required to write an initial review of their assigned papers. Afterwards, the discussion phase starts. Here, all reviewers of a paper are shown all the reviews other people wrote for the same paper. They can then alter their review based on the information they have seen. This discussion phase continues for multiple turns until the PC chair ends the phase.

3.1 Workflow Language

Workflows are defined over signatures $\Sigma = (S, C, \mathcal{R}, ar)$, with $\mathcal{R} = \mathcal{R}_{wf} \uplus \mathcal{R}_{high}$. Symbols in $\mathcal{R}_{wf}$ denote *workflow relations*, which are updatable. Symbols in $\mathcal{R}_{high}$ denote non-updatable relations that contain *high input* (i.e. input containing potentially confidential

[2] The decidable FOL fragments in [1, 27] are larger than the projection of the $\exists^*$FOLTL fragment to sorted FOL, as they also consider function symbols.

% PC members may declare conflicts
(b_1) **forall** $x{:}A, p{:}P$ **may**. $true \rightarrow Conf \mathrel{+}= (x, p)$
% PC members are assigned to papers
(b_2) **forall** $x{:}A, p{:}P$ **may**. $\neg Conf(x, p) \rightarrow Assign \mathrel{+}= (x, p)$
% PC members write reviews for papers
(b_3) **forall** $x{:}A, p{:}P, r{:}R$.
$Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$
% PC members discuss about the papers
loop (*) {
% PC members read all other reviews
(b_4) **forall** $x{:}A, y{:}A, p{:}P, r{:}R$.
$Assign(x, p) \wedge Review(y, p, r) \rightarrow Read \mathrel{+}= (x, y, p, r)$
% PC members can rethink their reviews
(b_5) **forall** $x{:}A, p{:}P, r{:}R$ **may**.
$Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$ }

Figure 1: EasyChair-like workflow.

$$
\begin{array}{lll}
w & ::= & block \mid block\ w \\
 & & \mid \textbf{loop (*)}\ \{w\} \mid \textbf{choose}\ w\ \textbf{or}\ w \\
block & ::= & \textbf{forall}\ x_1{:}s_1, \ldots, x_k{:}s_k.\ stmts \\
 & & \mid \textbf{forall}\ x_1{:}s_1, \ldots, x_k{:}s_k.\ \textbf{may}\ stmts \\
stmts & ::= & stmt \mid stmt; stmts \\
stmt & ::= & \theta \rightarrow R \mathrel{+}= (u_1, \ldots, u_n) \\
 & & \mid \theta \rightarrow R \mathrel{-}= (u_1, \ldots, u_n)
\end{array}
$$

Figure 2: Definition of Workflows

data) to the workflow. For instance, in Example 3.1 we have $\mathcal{R}_{wf} = \{Conf, Assign, Review, Read\}$ and $\mathcal{R}_{high} = \{Oracle\}$.

Workflows w over a signature $\Sigma = (S, C, \mathcal{R}, ar)$ are defined by the grammar given in Figure 2, where $s_1, \ldots, s_k$ with $k \geq 0$ range over sorts in S, x_i ranges over variables in $\mathcal{V}_{s_i}$ for each i, $u_1, \ldots, u_n$ with $n \geq 0$ range over terms in $\mathcal{V} \cup C$, R ranges over predicate symbols in $\mathcal{R}$, and θ ranges over first-order formulas over the signature Σ. For a statement $\theta \rightarrow R\pm=\bar{u}$, we require that $R \in \mathcal{R}_{wf}$, $|\bar{u}| = |ar(R)|$, and $fv(\theta) \cup fv(\bar{u}) \subseteq \bar{x}$, where $\bar{x}$ is the sequence of variables appearing in the **forall** construct of the block that contains the statement. We only consider well-sorted workflows.

3.2 Semantics

We will now give the semantics of workflows directly using FOLTL.

Formalization of the control flow. Given a workflow w, we consider its control flow graph (CFG), defined as expected. We add a node n_{end} to the CFG with only a single, looping, outgoing edge, encoding a fictitious new last block that can be reached after the original last block of the workflow. This new node is used to encode a finite (terminated) execution of the workflow by an infinite trace, where the last workflow state is stuttered. Any infinite path through the graph represents thus an execution of the workflow.

All edges are labelled with blocks. Note that for each block there is a unique edge that is labeled with that block. Edges not corresponding to a workflow block are labeled with a distinguished label *id*, which is assumed to represent an empty block without

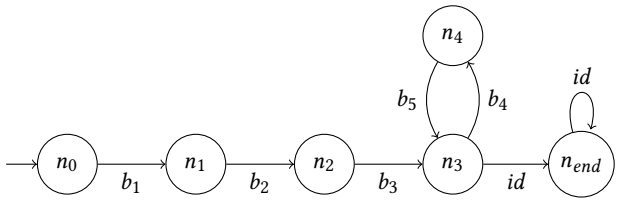

Figure 3: CFG of the workflow in Example 3.1.

variables and statements. As an illustration of the CFGs we consider, Fig. 3 depicts the CFG of the workflow in Example 3.1.

Let (V, E) be the CFG of workflow w. We abuse notation, and we use the proposition (i.e. nullary predicate) n to express whether the workflow is in node $n \in V$. We denote by $\mathcal{R}_{cfg}$ the set of these predicate symbols. The transition relation of the workflow is then expressed by the formula:

$$\mathtt{cfg}(w) := \mathsf{G}\left(\bigwedge_{n \in V} n \rightarrow \mathsf{X}\left(\bigvee_{n' \in succ(n)} n'\right)\right)$$

where $succ(n)$ is the set of the successors of the node n in the CFG. Furthermore, the workflow can never be in two states at once:

$$\mathtt{sanity}(w) := \mathsf{G}\left(\bigwedge_{n, n' \in V, n \neq n'} \neg(n \wedge n')\right)$$

Workflow loops can but need not terminate, and thus the same holds for workflow executions. A terminating behavior could be imposed by requiring that the node n_{end} is eventually reached, using the formula $\mathsf{F}\, n_{end}$. We do not impose this requirement.

Initial state. Every workflow executes sequentially, thus starting at node n_0 of the workflows CFG, where n_0 is the entry point in the CFG. There all relations in $\mathcal{R}_{wf}$ are empty. Formally, this initial condition is expressed by the following formula:

$$\mathtt{init}(w) := \left(n_0 \wedge \bigwedge_{n \in V \setminus \{n_0\}} \neg n\right) \wedge \bigwedge_{R \in \mathcal{R}_{wf}} \forall \bar{y}.\ \neg R(\bar{y})$$

Block execution. The basic step of our workflow language, which determines the transition from the current time point to the next time point, is the execution of one block. As in [16], we formalize this execution by characterizing with an FOLTL formula the interpretation of a predicate at the next time point based on the interpretation of the relations at the current time point.

For every block b and every relation symbol $R \in \mathcal{R}_{wf}$, we construct a formula $\Phi_{b,R}(\bar{y})$ so that $R(\bar{y})$ holds after execution of block b iff $\Phi_{b,R}(\bar{y})$ holds before the execution of b, with $|\bar{y}| = |ar(R)|$. The semantics of a block is then represented by the following formula:

$$\mathtt{exec}_w(b) := (n \wedge \mathsf{X}\, n') \rightarrow \bigwedge_{R \in \mathcal{R}_{wf}} \left(\forall \bar{y}.(\mathsf{X}\, R(\bar{y})) \leftrightarrow \Phi_{b,R}(\bar{y})\right)$$

where $(n, n') \in E$ is the edge in the CFG with label b.

For defining the formulas $\Phi_{b,R}$, we consider first a **may** block b, of the form **forall** $\bar{x}{:}\bar{s}$ **may** *stmts*. We assume, for simplicity, that each $R \in \mathcal{R}_{wf}$ is updated at most once in a block (i.e. it occurs at most once in a statement of block b on the right-hand side of $\rightarrow$).

For each $R \in \mathcal{R}_{wf}$, we let $\Phi_{b,R}(\bar{y}) :=$

$$\begin{cases} R(\bar{y}) & \text{if } R \text{ not updated} \\ R(\bar{y}) \vee \exists \bar{x}{:}\,\bar{s}.\ \theta \wedge \mathit{Choice}_i(\bar{x}) \wedge (\bar{y} = \bar{u}) & \text{if } \bar{u} \text{ added to } R \\ R(\bar{y}) \wedge \neg\big(\exists \bar{x}{:}\,\bar{s}.\ \theta \wedge \mathit{Choice}_i(\bar{x}) \wedge (\bar{y} = \bar{u})\big) & \text{if } \bar{u} \text{ deleted from } R \end{cases}$$

where block b's statement that updates R has the form $\theta \to R \pm\!= \bar{u}$, i is the index of the block b in the linearisation of the workflow, and it is assumed that $\bar{x} \cap \bar{y} = \emptyset$. The definition of $\Phi_{b,R}$ when b is a non-**may** block is similar, except that the $\mathit{Choice}_i(\bar{x})$ conjuncts are omitted.

If the same relation is modified multiple times in a block, the updates take place sequentially. This is expressed by building the formulas $\Phi_{b,R}$ inductively, similarly as above: $\Phi_{b,R}$ up to statement j is obtained by replacing $R(\bar{y})$ with $\Phi_{b,R}$ up to statement $j-1$. We omit the precise formalization.

We denote by $\mathcal{R}_{low}$ the Choice_i predicate symbols used in the $\Phi_{b,R}$ formulas. These symbols denote relations that contain *low input* (i.e. non-confidential input) to the workflow. For instance, in Example 3.1 we have $\mathcal{R}_{low} = \{\mathit{Choice}_1, \mathit{Choice}_2, \mathit{Choice}_5\}$.

The semantics of the workflow execution is then captured by the following formula:

$$\mathtt{exec}(w) := \mathsf{G} \bigwedge_{b \in \mathit{blocks}(w)} \mathtt{exec}_w(b).$$

where $\mathit{blocks}(w)$ is the set of w's blocks.

Example 3.2. The execution semantics of the block (b_2) of the workflow from Example 3.1 is given by the following formula:

$$\begin{aligned} &(n_1 \wedge \mathsf{X}\, n_2) \to \\ &\quad \Big(\forall y_1, y_2.\ \mathsf{X}\, \mathit{Assign}(y_1, y_2) \leftrightarrow \mathit{Assign}(y_1, y_2) \vee \\ &\qquad (\exists x, p.\ \mathit{Choice}_1(x, p) \wedge \neg \mathit{Conf}(x, p) \wedge y_1 = x \wedge y_2 = p)\Big) \wedge \\ &\quad \Big(\forall y_1, y_2.\ \mathsf{X}\, \mathit{Conf}(y_1, y_2) \leftrightarrow \mathit{Conf}(y_1, y_2)\Big) \wedge \ldots \end{aligned}$$

The first conjunct of the above consequent can be rewritten into the logically equivalent formula

$$\forall x, p.\ \mathsf{X}\, \mathit{Assign}(x, p) \leftrightarrow \mathit{Assign}(x, p) \vee \mathit{Choice}_1(x, p) \wedge \neg \mathit{Conf}(x, p)$$

by substituting x and p by y_1 and y_2 respectively, and then renaming y_1 and y_2 back to x and p. We note that this formula matches well the syntax of the block (b_2). The mentioned simplification cannot be performed in general, but only for a class of workflows, see Section 3.3.

We note that a **forall** $\bar{x}$ **may** block with statements of the form $\theta_i \to R_i \pm\!= \bar{u}_i$, can be seen as an abbreviation of a non-**may** block with statements of the form $\mathit{Choice}(\bar{x}) \wedge \theta_i \to R_i \pm\!= \bar{u}_i$, for some predicate symbol $\mathit{Choice} \notin \mathcal{R}_{wf}$. Note also that for an atom $\mathit{Oracle}(\bar{t})$ occurring in some guard θ_i, the arity of Oracle need not be $|\bar{x}|$. We use the abbreviated **may** form to emphasize the subtle differences between the two kinds of non-workflow relations.

Summary. The complete specification $\mathtt{wf}(w)$ of the workflow is a conjunction of the several parts described previously — the control flow graph, the initial state, and the semantics of the transitions between time points.

$$\mathtt{wf}(w) := \mathtt{cfg}(w) \wedge \mathtt{sanity}(w) \wedge \mathtt{init}(w) \wedge \mathtt{exec}(w)$$

Note that the formula $\mathtt{wf}(w)$ is expressed over the signature Σ' obtained from Σ by extending it with relation symbols $n \in \mathcal{R}_{cfg}$ and $\mathit{Choice}_i \in \mathcal{R}_{low}$.

For any given workflow w over a signature Σ, its semantics $[\![w]\!]$ consists of all temporal structures $\bar{\mathcal{S}}$ over Σ' that satisfy $\mathtt{wf}(w)$. A workflow w *satisfies* a closed FOLTL formula φ, denoted $w \models \varphi$, iff $\bar{\mathcal{S}}, \nu \models \varphi$ for any $\bar{\mathcal{S}} \in [\![w]\!]$, and any valuation ν. We have:

Theorem 3.3. *Given a workflow w, a* FOLTL *formula φ_w can be built in polynomial time so that for every* FOLTL *formula φ, it holds that $w \models \varphi$ iff $\varphi_w \to \varphi$ is valid.*

In fact, as such φ_w we may choose the formula $\mathtt{wf}(w)$.

3.3 Non-omitting Workflows

We call a workflow *non-omitting* iff for each of its blocks

$$\begin{aligned} &\textbf{forall } \bar{x}{:}\,\bar{s}\ [\textbf{may}] \\ &\quad \theta_1 \to R_1 \pm\!= \bar{u}_1; \\ &\quad \ldots \\ &\quad \theta_n \to R_n \pm\!= \bar{u}_n \end{aligned}$$

we have $fv(\bar{u}_i) = \bar{x}$ and θ_i is quantifier-free, for each $i \in \{1, \ldots, n\}$.

For a non-omitting workflow w, we can replace all existentially quantified variables inside the $\Phi_{b,R}$ formulas by their respective values, and remove the existential quantifiers. Thus, for any block b, as all guards of b are quantifier-free, $\Phi_{b,R}$ becomes quantifier-free, for all $R \in \mathcal{R}_{wf}$. It follows that the formula $\mathtt{wf}(w)$ can be brought into the $\exists^*$FOLTL fragment. Note that the thus simplified $\mathtt{wf}(w)$ formula contains no existential quantifiers. Furthermore, all its universal quantifiers are either not under a temporal operator (in the case of the $\mathtt{init}(w)$ subformula) or under the G temporal operator (in the case of the $\mathtt{exec}(w)$ subformula). Therefore the simplified $\mathtt{wf}(w)$ formula can be put in prenex normal form having a quantifier prefix consisting of only universal quantifiers. As a side remark, this means that $\neg\mathtt{wf}(w)$ can also be brought into $\exists^*$FOLTL.

Theorem 3.4. *It is decidable for a non-omitting workflow w and a formula φ in* $\exists^*$FOLTL *whether or not $w \models \neg\varphi$ holds.*

This means that if the set of all *bad* behaviors can be expressed by a formula φ in $\exists^*$FOLTL, then absence of bad behaviors can be checked for non-omitting workflows. The theorem follows from Theorems 3.3 and 2.1. Indeed, it is sufficient to check whether $\mathtt{wf}(w) \wedge \varphi$ is unsatisfiable. This can be done, since both conjuncts can be brought into $\exists^*$FOLTL, and thus the conjunction itself too.

The following theorem shows that the decidability result from Theorem 3.4 cannot be lifted to arbitrary workflows.

Theorem 3.5. *It is undecidable for a workflow w and a formula φ in* $\exists^*$FOLTL *whether or not $w \models \neg\varphi$ holds.*

Proof. We prove the theorem by reducing the *periodic tiling problem* to our workflow setting. The tiling problem was first mentioned in [31] and has first been shown undecidable by Berger in [6]. Its closely related variant, the periodic tiling problems has also been proven undecidable by multiple authors — for an overview see [20]. We now briefly recall the definition of the problem.

Given a set of k tile types $T = \{T_i \mid 0 \le i < k\}$ as well as horizontal and vertical compatibility relations x-comp $\subseteq T \times T$ and y-comp $\subseteq T \times T$, a *tiling* is a function $f(x, y) : \mathbb{N} \times \mathbb{N} \to T$

such that whenever two tiles are adjacent, they have to respect the compatibility relations:

$$\forall x, y.\ x\text{-comp}(f(x,y), f(x+1,y)),$$
$$\forall x, y.\ y\text{-comp}(f(x,y), f(x,y+1)).$$

A tiling is *periodic* if there exist horizontal and vertical periods p_x and p_y, such that

$$\forall x, y.\ f(x,y) = f(x+p_x, y),$$
$$\forall x, y.\ f(x,y) = f(x, y+p_y).$$

The periodic tiling problem is to find out for a given set of tile types and its compatibility relations, if there exists a periodic tiling.[3]

We will now proceed to show how to encode this problem in our workflow setting. We note that to find a periodic tiling, it is enough to find the periods p_x, p_y, and the values $f(x,y)$ for $0 \le x < p_x$ and $0 \le y < p_y$, such that they are also compatible at borders:

$$\forall y.\ x\text{-comp}(f(p_x,y), f(0,y)),$$
$$\forall x.\ y\text{-comp}(f(x,p_y), f(x,0)).$$

We thus see a periodic tiling as a table with rows referring to points on the y-axis and columns referring to points on the x-axis.

We build next a workflow w and a formula φ such that $w \models \varphi$ iff there is a periodic tiling for $(T, x\text{-comp}, y\text{-comp})$. We use the following signature:

$$\Sigma = \Big(\{A\}, \{a_{first}, a_{last}\}, \{Q, Adj, Reach, T'_0, \ldots, T'_{k-1}\}, ar\Big)$$

Intuitively, time points refer to the rows of the tiling, while agents refer to its columns. We explain next the role of the constant and relation symbols. The k unary relations T'_i, with $0 \le i < k$, are used to encode the tiling function as follows: if $T'_i(a_j)$ holds at time point t, for some particular agent a_j, then the tiling function is $f(t,j) = T_i$. How the agent a_j is determined is explained later. There are two constant agents a_{first} and a_{last} which are used to name the first and last row of the tiling. The nullary relation Q encodes the last column of the tiling. The predicate $Adj(a, a')$ expresses that the row named by a' is directly below the row named by a. Only *Reach* is a workflow relation; thus, initially (i.e. at time point 0) it is empty. There is a single sort, the agent sort A.

We let w be the following workflow. It is used to compute all reachable parts of the adjacency relation *Adj* starting from the initial agent a_{first}:

forall. $true \to Reach \mathrel{+}= (a_{first})$
loop (*)
  forall a, a'. $Reach(a) \wedge Adj(a, a') \to Reach \mathrel{+}= (a')$

To encode the rest of the tiling requirements, we use a conjunction of $\exists^*$FOLTL formulas, where i, j implicitly range over the elements in $\{0, \ldots, k-1\}$:

All agents always have exactly one tile assigned at each point in time (Eqs. (1) and (2)).

$$\mathsf{G}\,\forall a.\ \bigvee_i T'_i(a) \tag{1}$$

$$\mathsf{G}\,\forall a.\ \bigwedge_{i \ne j} T'_i(a) \to \neg T'_j(a) \tag{2}$$

A time point will be reached state where Q holds (Eq. (3)) and it will only hold once (Eq. (4)).

$$\mathsf{X}\,\mathsf{F}\,Q \tag{3}$$

$$\mathsf{G}\,(Q \to \mathsf{X}\,\mathsf{G}\,\neg Q) \tag{4}$$

Two adjacent time points need to be assigned x-compatible tiles (Eq. (5)). Also, the right border of the tiling should be x-compatible to the left, i.e. the time point where Q holds should be compatible to the starting time point (Eq. (6)).

$$\forall a.\ \mathsf{G}\,\Big(\bigvee_{i,j:\, x\text{-comp}(T_i,T_j)} T'_i(a) \wedge \mathsf{X}\,T'_j(a)\Big) \tag{5}$$

$$\forall a.\ \bigvee_{i,j:\, x\text{-comp}(T_i,T_j)} T'_j(a) \wedge \mathsf{F}\,\Big(Q \wedge T'_i(a)\Big) \tag{6}$$

Two adjacent agents need to be assigned y-compatible tiles (Eq. (7)). The last agent should be reachable from the first via *Adj* relations. We cannot express this fact in pure $\exists^*$FOLTL, so we will use the relation *Reach* computed by the workflow (Eq. (8)). The last agent should also be y-compatible to the first (Eq. (9)).

$$\mathsf{G}\,\forall a, a'.\ (\mathsf{F}\,Adj(a,a')) \to \bigvee_{i,j:\, y\text{-comp}(T_i,T_j)} T'_i(a) \wedge T'_j(a') \tag{7}$$

$$\mathsf{F}\,Reach(a_{last}) \tag{8}$$

$$\mathsf{G}\,\Big(\bigvee_{i,j:\, y\text{-comp}(T_i,T_j)} T'_i(a_{last}) \wedge T'_j(a_{first})\Big) \tag{9}$$

Let φ be the conjunction of Eqs. (1) to (9). Note that φ can be brought in $\exists^*$FOLTL. We show next that $w \models \varphi$ iff there is a periodic tiling for $(T, x\text{-comp}, y\text{-comp})$.

Let $\bar{\mathcal{S}} \in [\![w]\!]$ such that $\bar{\mathcal{S}} \models \varphi$. We construct a tiling as follows. As $\bar{\mathcal{S}}$ satisfies the formula (8) and the formulas encoding the first and second blocks of the workflow, it follows that there is a sequence $(t_0, \ldots, t_n)$ of time points with $n > 0$ and $t_0 = 0$, and a sequence $(a_0, \ldots, a_n)$ of elements of the universe such that $a_0 = a_{first}$, $a_n = a_{last}$, $Reach(a_i)$ holds at time point t_i, for all i with $0 \le i \le n$, and $Adj(a_i, a_{i+1})$ holds at time point t_i, for all i with $0 \le i < n$. Then, we set p_x to the time point where Q holds and p_y to n. For $0 \le t < p_x$ and $0 \le j < p_y$, let $f(t,j)$ be T_i iff $T'_i(a_j)$ holds at time point t. It is easy to see that f satisfies the compatibility relations and that any given tiling can be transformed into a model of φ. □

4 HYPERPROPERTIES

In this section, we show how to formalize and verify security properties of workflows. We focus on non-interference properties [17], which are hyperproperties [10]. To specify such properties we use the first-order extension of HyperLTL [9] presented in [16]. HyperLTL can relate multiple traces and it is thus well suited to express not only trace properties, but also hyperproperties. The first-order

[3] The original formulation used just a single period p in both directions. We use independent periods to have less complicated constructions. We note that given a periodic tiling t with periods p_x and p_y it is easy to construct a periodic tiling t' with $p'_x = p'_y = (p_x * p_y)$. The original problem also did not consider compatibility relations, but edges of the same color. Again this makes our constructions easier and is easily transformed into a solution of the original setting.

extension is needed in the presence of an unbounded number of agents. Furthermore it allows for more fine-grained policies.

4.1 HyperFOLTL

For presenting the syntax and semantics of the logic, we follow [16].

Syntax. Let $\Sigma = (S, C, \mathcal{R}, ar)$ be a signature, and let Π be a set of *trace variables* disjoint from the set $\mathcal{V}$ of first-order variables. Let $\mathcal{R}_\Pi = \{R_\pi \mid R \in \mathcal{R}, \pi \in \Pi\}$ and $\Sigma' = (S, C, \mathcal{R}_\Pi, ar')$ be the signature with $ar'(R_\pi) = ar(R)$, for any $R \in \mathcal{R}$ and $\pi \in \Pi$.

HyperFOLTL extends FOLTL as follows. HyperFOLTL *formulas* over Σ and Π are then generated by the following grammar:

$$\psi ::= \exists \pi.\, \psi \mid \neg\psi \mid \varphi$$

where $\pi \in \Pi$ is a trace variable and φ is a FOLTL formula over Σ'. Universal trace quantification is defined as $\forall\pi.\psi := \neg\exists\pi.\neg\psi$. HyperFOLTL formulas thus start with a prefix of trace quantifiers consisting of at least one quantifier and then continue with a subformula that contains only first-order quantifiers, no trace quantifiers. As for FOLTL, a formula without free first-order and trace variables is called *closed*.

Semantics. The semantics of a HyperFOLTL formula ψ is given with respect to a set $\mathcal{T}$ of temporal structures, a valuation $\alpha : \mathcal{V} \to U$ of the first-order variables, and a valuation $\beta : \Pi \to \mathcal{T}$ of the trace variables. The *satisfaction* of a HyperFOLTL formula ψ, denoted by $\mathcal{T}, \alpha, \beta \models \psi$, is then defined as follows:

$$\begin{array}{lll} \mathcal{T}, \alpha, \beta \models \exists\pi.\, \psi & \text{iff} & \mathcal{T}, \alpha, \beta[\pi \mapsto t] \models \psi, \text{ for some } t \in \mathcal{T}, \\ \mathcal{T}, \alpha, \beta \models \neg\psi & \text{iff} & \mathcal{T}, \alpha, \beta \not\models \psi, \\ \mathcal{T}, \alpha, \beta \models \varphi & \text{iff} & \bar{\mathcal{S}}, \alpha \models \varphi, \end{array}$$

where ψ is an HyperFOLTL formula, φ is an FOLTL formula, and the temporal structure $\bar{\mathcal{S}}$ is such that for all $R \in \mathcal{R}$, $i \in \mathbb{N}$, and, $\pi \in \Pi$, the interpretation $R_\pi^{\mathcal{S}_i}$ is $R^{\beta(\pi)(i)}$ if π in the domain of β, and $\emptyset$ otherwise.

A HyperFOLTL formula ψ is *satisfiable* iff there exists a set $\mathcal{T}$ of temporal structures and valuations α and β s.t. $\mathcal{T}, \alpha, \beta \models \psi$.

Example 4.1. Observational determinism [33] of programs can be formalized by the following HyperFOLTL formula

$$\forall\pi, \pi'.\, (\mathrm{G}\, \forall x.\, I_\pi(x) \leftrightarrow I_{\pi'}(x)) \rightarrow (\mathrm{G}\, \forall y.\, O_\pi(y) \leftrightarrow O_{\pi'}(y)),$$

where $I(x)$ denotes that x is a low input to the program, while $O(y)$ denotes that y is a low output. The inputs and outputs are classified as low or high with respect to the clearance level of some particular user. The formula states that, on any two program executions, if the low inputs are always the same, then the low outputs are also always the same. That is, from a low user point of view, the observable behavior of the program is only determined by its inputs.

We will adapt this non-interference notion to the workflow setting in Section 4.2. We refer to [9] for the formalization in HyperLTL of other hyperproperties.

Decidability. We will consider the fragment of HyperFOLTL, named $\exists^*_\pi\forall^*_\pi\exists^*$FOLTL, that consists of all formulas of the form $\exists\pi_1, \ldots \pi_k.\, \forall\pi'_1 \ldots \pi'_\ell.\, \varphi$ with $k \geq 0$, $\ell \geq 0$, and φ an FOLTL formula in $\exists^*$FOLTL.

We first remark that by seeing trace variables as first-order variables of a new sort T — the trace sort, HyperFOLTL formulas can be faithfully encoded by FOLTL formulas. By this we mean that, for any closed HyperFOLTL formula ψ, there is a closed FOLTL formula φ such that we can translate models of ψ into models of φ and vice-versa. The formula φ is obtained by replacing trace quantification $Q\pi$ to first-order quantification $Q\pi{:}\,T$, for $Q \in \{\exists, \forall\}$, and predicates $R_\pi(\bar{u})$ with predicates $R'(\pi, \bar{u})$. Note that ψ and φ are formulas over slightly different signatures. The translation between models is straightforward. For instance, if $\bar{\mathcal{S}}$ is a temporal structure that satisfies φ, then the corresponding set $\mathcal{T}$ of temporal structures that satisfies ψ consists of temporal structures obtained by projecting a predicate's interpretation on the predicate's non-trace arguments, for each of the values of the trace universe U_T of $\bar{\mathcal{S}}$, i.e. $\mathcal{T} = \{\bar{\mathcal{S}}_t \mid t \in U_T\}$ and $R^{\mathcal{S}_{t,i}} = \{\bar{a} \mid (t, \bar{a}) \in R'^{\bar{\mathcal{S}}_i}\}$, for each $R \in \mathcal{R}$, $t \in U_T$, and $i \in \mathbb{N}$.

As a consequence of the previous discussion, and as a corollary of Theorem 2.1, we obtain the following results.

Theorem 4.2. *The following statements hold.*

(1) *Every* HyperFOLTL *formula can be translated into an equisatisfiable* FOLTL *formula.*
(2) *Satisfiability of formulas in* $\exists^*_\pi\forall^*_\pi\exists^*$FOLTL *is decidable.*

Workflow satisfaction. A workflow w *satisfies* a closed formula HyperFOLTL ψ, denoted $w \models \psi$, iff $[\![w]\!], \alpha, \beta \models \psi$ for the empty assignments α and β.

Theorem 4.3. *Let w be a workflow and ψ a* HyperFOLTL *formula. Then the following statements hold.*

(1) *An* FOLTL *formula ψ' can be constructed in polynomial time so that $w \models \neg\psi$ iff ψ' is unsatisfiable.*
(2) *If w is non-omitting and ψ is in $\exists^*_\pi\forall^*_\pi\exists^*$*FOLTL*, then it is decidable whether or not $w \models \neg\psi$ holds.*

Proof. Assume ψ has the form $Q_1\pi_1 \ldots Q_k\pi_k.\, \varphi$, where the trace quantifiers are partitioned into a set E of existential quantifiers and a set A of universal quantifiers. Then $w \models \neg\psi$ is equivalent with the validity of the following HyperFOLTL formula

$$\bar{Q}_1\pi_1 \ldots \bar{Q}_k\pi_k.\, \Big(\bigwedge_{\bar{Q}_i \in E} \mathtt{wf}(w)_{\pi_i}\Big) \wedge \Big(\Big(\bigwedge_{\bar{Q}_i \in A} \mathtt{wf}(w)_{\pi_i}\Big) \rightarrow \neg\varphi\Big),$$

where $\mathtt{wf}(w)_\pi$ is $\mathtt{wf}(w)$ with each predicate symbol R replaced by the predicate symbol R_π, and $\bar{Q}$ is $\exists$ if Q is $\forall$ and vice-versa. The formula ψ' is then the FOLTL encoding of the following HyperFOLTL formula

$$Q_1\pi_1 \ldots Q_k\pi_k.\, \Big(\bigwedge_{Q_i \in A} \mathtt{wf}(w)_{\pi_i}\Big) \rightarrow \Big(\Big(\bigwedge_{Q_i \in E} \mathtt{wf}(w)_{\pi_i}\Big) \wedge \varphi\Big).$$

From Theorem 4.2, to prove the second statement, it is sufficient to show that the previous HyperFOLTL formula, which we call ψ_1, can be brought in the $\exists^*_\pi\forall^*_\pi\exists^*$FOLTL. By assumption, we have that the trace quantifier prefix of ψ_1 is of the form $\exists^*_\pi\forall^*_\pi$ and that φ is in the $\exists^*$FOLTL fragment. Also, since w is non-omitting, then both $\mathtt{wf}(w)$ and $\neg\mathtt{wf}(w)$ can be brought in the $\exists^*$FOLTL fragment, as remarked in Section 3.3. Thus all conjuncts in the following FOLTL formula can be brought in the $\exists^*$FOLTL fragment

$$\Big(\bigwedge_{Q_i \in A} \neg\mathtt{wf}(w)_{\pi_i}\Big) \vee \Big(\Big(\bigwedge_{Q_i \in E} \mathtt{wf}(w)_{\pi_i}\Big) \wedge \varphi\Big)$$

This means that the formula itself can be put into $\exists^*$FOLTL and thus ψ_1 can be brought into $\exists^*_\pi \forall^*_\pi \exists^*$FOLTL. □

4.2 Non-interference in workflows

As we have defined it, the workflow keeps track of the state of all relations of all agents. However, security policies are meant to allow access to classified information to just some of the users of the system while denying it to others. For this, we need to specify how an agent interacts with the workflow and reason about his knowledge and possible interactions with the system.

In the running example, members of the PC can use a conference management system to specify conflicts, read the reviews that other members have provided, provide their own reviews, etc. As an example property, we will formalize that no member of the PC gains any information about papers that he declared a conflict of interest with.

Following [16], we present a variant of *non-interference* suitable for these properties on workflows by adapting the notion of observational determinism from Example 4.1 to explicitly take into account the knowledge and behavior of participating agents.

Non-interference in general is a strong specification of the valid information flows in a system. It uses a classification of all inputs and outputs to a system into "high" security and "low" security inputs [17]. In our setting, these notions of input and output are specific to an agent and his interactions with the workflow. input to model a's interactions with the workflow. We call a workflow *non-interferent*, iff for any agent a, his observations do not depend on the inputs which are "high" for a in any way.

Agent Model. It has been observed in [16], that non-interference in workflows can only reasonably be argued about, if meaningful assumptions on the behavior of agents are provided.

In order to specify such assumptions, we make the convention that in any relation recording an agent's knowledge or interaction, this agent appears in the first argument of the relation. Formally, we classify all sorts into agent sorts and data sorts. Moreover, we require that the arity $(s_1, s_2, \dots)$ of every relation $R \in \mathcal{R}_{wf} \cup \mathcal{R}_{low} \cup \mathcal{R}_{high}$ is non-nullary and is such that s_1 is an agent sort. This restriction, while not strictly necessary, allows us to present the results in this section in a much cleaner way.

An agent provides observable input to the workflow system by choosing to execute (or to not execute) **may**-blocks for specific data. Such input is low input, formalized through the predicates $Choice \in \mathcal{R}_{low}$. The property that at a given time point, all low inputs for a given agent a are equal on traces π, π' is formalized as:

$$same\_low\_inputs_{\pi,\pi'}(a) := \bigwedge_{Choice \in \mathcal{R}_{low}} \left(\forall \bar{x}.\, Choice_\pi(a, \bar{x}) \leftrightarrow Choice_{\pi'}(a, \bar{x})\right),$$

where, for each *Choice* predicate, the sequence $\bar{x}$ has the same length as its arity minus 1. Input provided by the environment is considered high input. It is formalized through predicates $Oracle \in \mathcal{R}_{high}$.

An agent can observe all tuples in which it is mentioned in the first argument. The property that, at a time point, all observations of a given agent a are the same on two given traces π and π' is formalized by the following formula:

$$same\_observations_{\pi,\pi'}(a) := \bigwedge_{R \in \mathcal{R}_{wf}} (\forall \bar{x}.\, R_\pi(a, \bar{x}) \leftrightarrow R_{\pi'}(a, \bar{x}))$$

Agent Behavior. The behavior of the workflow as seen by one of the agents, depends on the actions of all other agents. If agents have the power to behave arbitrarily, there will be spurious counterexample traces to confidentiality where an agent chooses to let his actions depend on confidential data — which he could not even access. Here, we consider two meaningful agent models which restrict the behavior of agents across different executions.

The simpler agent model considers *stubborn* agents. An agent is called stubborn if, even when told information that is confidential to another agent, he will not choose his actions depending on this information. Thus, anyone observing the behavior of a stubborn agent will not be able to conclude anything about confidential data. Technically, this amounts to saying that his choices are independent of the chosen trace. For a pair of traces π, π', the behavior of a stubborn agent is therefore specified in HyperFOLTL by the following formula:

$$stubborn_{\pi,\pi'}(a) := \mathrm{G}\ same\_low\_inputs_{\pi,\pi'}(a)$$

A more intricate model of agent behavior considers *causal* agents. An agent is called causal if his actions may depend on his observations. As a result, a causal agent can subtly change his behavior depending on the data that he gained access to. As an example, a causal agent could indicate the acceptance of a paper to someone else either by explicitly telling it to someone or by commenting to another paper or refraining from it. For a pair of traces π, π', this behavior is specified in HyperFOLTL by the following formula.

$$causal_{\pi,\pi'}(a) := same\_low\_inputs_{\pi,\pi'}(a)\ \mathrm{W}\ \neg same\_observations_{\pi,\pi'}(a)$$

We remark that the causal agent model subsumes the stubborn agent model and is less constraining on the behavior of the individual agents, which leads to more intricate information flow violations.

We remark that the formula $\forall a.causal(\pi, \pi', a)$ is not expressible in $\exists^*$FOLTL, as it has an $\forall\exists$ quantifier structure. In case, however, that we consider a fixed upper bound on the number of causal agents, the corresponding formula is in the $\exists^*$FOLTL fragment. For instance for at most two agents, we can use the following formula:

$$\exists a_1, a_2.\, causal_{\pi,\pi'}(a_1) \wedge causal_{\pi,\pi'}(a_2)$$

Considering an upper bound on the number of causal agents is a realistic setting, as it allows to verify the system for attacks by coalitions up to a given size.

Declassification. In general, all external input data to the workflow, i.e. all relations in $\mathcal{R}_{high}$ are considered as high input. However, it often needs be possible that an agent can learn something about the high input data, depending on the scenario. This is also apparent in the conference management example. There, it is necessary for a reviewer to be able to read at least some reviews, namely, the reviews for papers he himself is assigned to — although reading these might be illegitimate for others.

To model declassification, we assume a formula φ_{Oracle} for each relation *Oracle* in $\mathcal{R}_{high}$. This formula encodes a declassification condition that describes which *Oracle* tuples represent declassified

Table 1: A counterexample to non-interference.

block	relation	π	π'
(b_1)	*Conf*	(a_1, p_1)	
(b_2)	*Assign*	$(a_1, p_2), (a_2, p_2), (a_2, p_1)$	
(b_3)	*Review*	(a_2, p_1, r_{21}) (a_2, p_2, r_{22})	 (a_2, p_2, r_{22})
(b_4)	*Read*	(a_1, a_2, p_2, r_{22}) (a_2, a_2, p_2, r_{22}) (a_2, a_2, p_1, r_{21})	(a_1, a_2, p_2, r_{22}) (a_2, a_2, p_2, r_{22})
(b_5)	*Review*	(a_2, p_2, r_{21})	
(b_4)	*Read*	(a_1, a_2, p_2, r_{21}) (a_2, a_2, p_2, r_{21})	

information, for any given agent. Initial high inputs for a therefore should only be equal on traces π, π' if they are declassified for agent a. Technically, this property is formalized by:

$$same\_declassified\_high\_inputs_{\pi,\pi'}(a) := \mathrm{G} \bigwedge_{Oracle \in \mathcal{R}_{high}} \forall \bar{y}. \left(\begin{array}{l} (\varphi_{Oracle,\pi}(a,\bar{y}) \vee \varphi_{Oracle,\pi'}(a,\bar{y})) \\ \rightarrow (Oracle_{\pi}(\bar{y}) \leftrightarrow Oracle_{\pi'}(\bar{y})) \end{array} \right)$$

By the notation $\varphi_{Oracle}(a, \bar{y})$ we mean that the free variables of the formula φ_{Oracle} are among the variables a and those in $\bar{y}$. For our running example, we use $\varphi_{Oracle}(a, x, p, r) := \neg Conf(a, p)$.

Control Flow. The structure of the control flow graph and the current position of the workflow (i.e. the state of all relations in $\mathcal{R}_{cfg}$) are considered as *low* input. This serves the intuition that the non-determinism in the workflow is resolved by some external control. For instance, the PC chair of the conference management system may terminate the submission loop. This assumption is formalized by the following formula:

$$same\_paths_{\pi,\pi'} := \mathrm{G} \bigwedge_{n \in \mathcal{R}_{cfg}} n_{\pi} \leftrightarrow n_{\pi'}$$

Putting it all together. Assume that there are at most $k \geq 0$ causal agents with all other agents being stubborn. *Non-interference with Declassification* is then expressed in HyperFOLTL by the following formula:

$$\begin{aligned} \forall \pi, \pi'. \Big(\exists a_1, \ldots, a_k. \Big(\bigwedge_{i=1}^{k} causal_{\pi,\pi'}(a_i) \Big) \wedge \\ \Big(\forall a. \big(\bigwedge_{i=1}^{k} a \neq a_i \big) \rightarrow stubborn_{\pi,\pi'}(a) \Big) \Big) \\ \wedge same\_paths_{\pi,\pi'} \\ \rightarrow \forall a.\, noninterferent_{\pi,\pi'}(a) \end{aligned}$$

where

$$\begin{aligned} noninterferent_{\pi,\pi'}(a) := \\ \Big(\big(\mathrm{G}\, same\_low\_inputs_{\pi,\pi'}(a) \big) \wedge \\ same\_declassified\_high\_inputs_{\pi,\pi'}(a) \Big) \\ \rightarrow \mathrm{G}\, same\_observations_{\pi,\pi'}(a). \end{aligned}$$

Example 4.4. Coming back to the workflow in Example 3.1, we check if the non-interference property holds.

When all agents are stubborn, we find that non-interference is satisfied for the given workflow. This result indicates that there is no way for any agent to learn confidential information without having a conspirator helping him.

The result is different when there is at least one causal agent. In this case we find the following counterexample: Assume two PC members a_1 and a_2 where a_1 is stubborn and a_2 is causal. The non-interference property is stated for a_1. There are two papers p_1 and p_2. First, a_1 declares a conflict with p_1, so in the rest of the workflow he should not be able to observe a difference between two executions of the workflow, regardless of which reviews p_1 receives. Both agents get assigned to p_2. In addition, a_2 gets assigned to p_1 and writes a review for it. At this point, a_2 can observe at least one review for p_1, so he can deviate his behavior on the two executions. The next step is the discussion phase. In the first step, a_2 reads all reviews of p_1. In the next step, a_2 adjusts his reviews of p_2 to mirror the reviews of p_1. Then, in the next iteration, a_1 will read the differing reviews of p_2 and learn about the result of p_1, the paper he initially declared a conflict with.

Table 1 formalizes the counterexample. It shows the tuples that are added to the updated relation after the execution of each block. Note that the workflow updates only one relation per block and there are no removals. The reviews for p_2 cannot differ (in the two traces) directly after the execution of the block (b_3) since the declassification condition states that tuples in *Oracle* can only differ when they are of the form (x, p_1, r). However, as a_2 can observe his own reviews for p_1, his choices can start to differ after (b_3) is executed; concretely, they will differ when block (b_5) is executed. In the last two rows, any value for r (except r_{22}) would result in a counter-example; we use r_{21} to suggest that a_2 could simply replace its review for p_2 with the review for p_1. This attack represents someone copy-pasting his review for the wrong paper into one of his reviews.

We note that for the given specification of the workflow, such an attack is unavoidable in "real life", as it can be performed also outside the workflow system. Concretely, a_2 can directly communicate the reviews for p_1 to a_1 through any communication channel, for instance by email. To combat this attack, the example should be changed to having disjunct reviewing groups — whenever a reviewer r is assigned to a paper p, no one else that has a conflict with the other assigned papers of r can be assigned to p.

4.3 Verification

As hinted in Section 4.1, given a non-omitting workflow w and an $\exists^*_\pi \forall^*_\pi \exists^*$FOLTL formula ψ denoting a set of bad behaviors, our approach for checking whether $w \models \neg\psi$ consists in checking the (un)satisfiability of the formula ψ' given in the proof of Theorem 4.3(1).

As an instance of this approach, we obtain that Non-interference with Declassification can be checked on non-omitting workflows.

Theorem 4.5. *For any non-omitting workflow, it is decidable to check whether it satisfies Non-interference with Declassification for a finite number of causal agents and an unbounded number of stubborn agents, as long as for each formula φ expressing a declassification*

condition, the negation normal form of $\neg\varphi$ contains no existential quantifier.

It is easy to check that the negation of non-interference can be brought into $\exists^*_\pi \forall^*_\pi \exists^*$FOLTL. Then, as w is non-omitting, the result follows directly from by Theorem 4.3(2).

In [16], the authors show that for workflows without loops, it is possible to check non-interference even when *all* agents behave in a causal way. This is no longer the case for workflows with loops:

Theorem 4.6. *The problem of checking for a given non-omitting workflow w wether it satisfies Non-interference with Declassification for an* unbounded *number of causal agents is undecidable, even if for all formulas φ expressing a declassification condition, the negation normal form of $\neg\varphi$ contains no existential quantifier.*

Proof. As in the proof for Theorem 3.5, we present a reduction from the periodic tiling problem. We will consider a workflow w over signature Σ with

$$\Sigma = \Big(\{A\}, \{a_{first}\}, \{Q, O, Obs, Adj, T'_0, \ldots, T'_{k-1}\}, ar\Big)$$

where A, a_{first}, Q, and $T'_0, \ldots, T'_{k-1}$ are as in proof for Theorem 3.5, and they fulfill the same purposes. The *Adj* symbols denote again a vertical adjacency relation, but here it is not filled with input data, but rather computed stepwise by the workflow. The relation denoted by O and *Obs* contain an initial secret that differs on both traces and spreads along the adjacency relation *Adj*. The symbols $Q, T'_0, \ldots, T'_{k-1}$ denote again high-input relations containing input data with a declassification of *true* (i.e. they are always equal in both traces).

We consider the following workflow:

forall . $Oracle(a_{first}) \rightarrow Obs \mathrel{+}= (a_{first})$
loop (*)
% Information flow from a to b
forall a, b **may**. $true \rightarrow Adj \mathrel{+}= (b, a)$
% Clear *Adj*
forall a, b. $Adj \mathrel{-}= (a, b)$

We add the rest of the tiling requirements to the declassification condition of O, so that there only is an information flow violation in case that all formulas hold.

As we again use time as the x-axis, we reuse Eqs. (1) to (6). We also use a_{last} as one representative agent of the bottom-most row, so we reuse Eq. (9) to specify that a_{last} is compatible to a_{first}. This time a_{last} is not part of the signature, but we will call the outermost agent of the non-interference condition a_{last}, so all declassification conditions can use the variable.
Two adjacent agents need to be assigned y-comp tiles (Eq. (10)).

$$\forall a, b.\ (\mathsf{F}\, Adj(b, a)) \rightarrow \mathsf{G} \bigvee_{y\text{-comp}(i,j)} T'_i(a) \wedge T'_j(b) \qquad (10)$$

The last agent should be reachable from the first via *Adj* relations. This is expressed by specifying that a_{last} can observe different tuples on traces π, π' (the non-interference property.). Let ψ be the conjunction of equations Eqs. (1) to (6) and (10). Let the declassification conditions be:

$$\varphi_{Oracle} = \neg\psi,\ \varphi_Q = true,\ \varphi_{T'_i} = true$$

We then verify the non-interference property

$$\forall \pi, \pi', a_{last}. \forall a \neq a_{last}.\ causal_{\pi,\pi'}(a) \rightarrow noninterferent(\pi, \pi', a_{last}) \qquad (11)$$

There exists a satisfying model for the negation of the property in Eq. (11) on w iff there is a periodic tiling for (T, x-comp, y-comp). If a_{last} observes different low outputs (tuples in *Adj*), either he is the same agent as a_{first} and y-compatible to himself or there exists a chain of causal agents spreading the tuples along *Adj* to a_{last}. Since a_{first} is the only one able to read *Oracle*, every possible differences can only originate in *Obs*. Thus, there is a chain of y-compatible causal agents a_i starting with a_{first} that reaches a_{last}. We can construct the tiling from any satisfying model in exactly the same way as in the proof of Theorem 3.5. □

5 EXPERIMENTAL EVALUATION

We have implemented our approach into the tool NIWO.[4] Our tool takes as input the specification of a workflow together with declassification conditions and the number of causal agents. From that, it generates a sorted FOLTL formula whose satisfiability is equivalent to the existence of a violation of the non-interference property. For non-omitting workflows, this formula is further compiled into an equi-satisfiable LTL formula to be checked by some of-the-shelf LTL satisfiability solver. Currently, we use Aalta [23] for that purpose. The NIWO tool is, to our knowledge, the first implementation of an automated verification approach for workflows.

5.1 Size of the formulas

We consider the structure and size of the formulas whose unsatisfiability is checked in order to establish whether non-interference holds for a given workflow. Such a formula is a conjunction with four conjuncts. One conjunct is a universally quantified formula that describes the semantics of the workflow (see Section 4.3). The next conjunct represents the agent model and it is an existentially quantified formula with the number of quantifiers matching the given upper bound on the number of causal agents. The third conjunct represent the assumption on the control flow and it is a propositional formula. The last conjunct describes the existence of a counterexample to the non-interference property and it is an existentially quantified formula. When considering only stubborn agents, for a workflow with observable relations of maximum arity n_s per sort s, the formula to be checked for unsatisfiability uses $\sum_s n_s$ existential quantifiers. Every existentially quantified variable adds one Skolem constant to the smallest universe (for its sort) that can be considered. As described in the proof of Theorem 2.1 universal quantification over a sort is translated into a conjunction over all Skolem constants of that sort.[5] Thus, the resulting encoding of the universally quantified conjunct is exponential in the number of existentially quantified variables of each sort. For every causal agent considered, $\sum_s n_s$ additional existential quantifiers are added to the formula. Since every causal agent adds multiple existential quantifiers, the resulting LTL formula can be orders of magnitude larger when considering multiple causal agents of the same sort.

[4] The source code together with all examples can be found on the authors' website.
[5] Note that, nevertheless, sorts may greatly reduce the number of conjuncts in comparison to the unsorted case.

5.2 Experiments

We evaluated NIWO on realistic example workflows, among these the example from the introduction, as well as synthetic examples to evaluate scalability issues.

We used several realistic examples. *Notebook* is an event-based model of a notebook-like data structure where several people can write messages, but everyone can only read his own data. It is proven safe by our implementation, even in the presence of causal agents. *Conference* is the example conference management from Example 3.1, where our implementation finds the counterexample described in Section 4.3. *Conference-acceptance* is a slight variation that forgoes reviews and replaces it by an acceptance relation. Since it is very similar to the initial conference example, we use it to showcase the impact of small changes to the workflow to the verification problem. *Conference-linear* is the motivating example used in [16]. It is a simpler version of the Conference example, which does not use loops, and exhibits a very similar attack. *University* is an example from a university environment where a professor writes down secret grading information for students. It takes at least 2 conspiring agents for a student to learn something about grades of other students.

Additionally, we used synthetic examples to illustrate the scalability of the approach in several dimensions. The *Fixed-Arity-X* examples show the behavior when increasing the number of relations. These cases contain X relations that are successive copies of each other starting from some secret input. The *Fixed-Arity-X-safe* examples are similar, but are devoid of counterexamples. The *Sorted-Increasing-Arity-X* and *Increasing-Arity-X* examples show the impact of using sorts. They contain X relations of arities $1, \ldots, X$, respectively. For every relation of arity n, a tuple containing the first $n-1$ variables has to be present in the relation with arity $n-1$. For the *Increasing-Arity* cases, all variables refer to the same sort, whereas in the *Sorted-Increasing-Arity* variant, every relation of arity n uses n different sorts. *causal-X* and *Sorted-causal-X* cases showcase the scalability with the number of causal agents that are part of the attack. These cases are set up in a way that a successful attack needs to consist of at least X causal agents.

5.3 Results

The results of the experiments are shown in Table 2. The first column describes the number of causal agents that are considered. All other participating agents are considered as stubborn as per Section 4.

The size of the workflow is the number of blocks the workflow consists of, not counting choice and loop constructs. The result is *safe* iff the LTL formula was proven *unsatisfiable* by Aalta and *unsafe* otherwise. The next column gives the sizes of the considered universes. For example, to show that *Conference* is safe with respect to one causal agent, it is enough to consider universes containing 4 reviewers, 2 papers and 2 reviews (one per paper), respectively. The universes' sizes are given as a tuple, for instance $(4, 2, 2)$. The size of both the FOLTL and LTL formulas is the number of nodes in the formulas abstract syntax tree. The last column is the time (in seconds) that it takes Aalta to check the satisfiability of the LTL formula (averaged over 10 runs). All experiments were carried out on a desktop machine using an Intel i7-3820 clocked at 3.60 GHz with 15.7 GiB of RAM and running Debian with a timeout of 20 minutes.

The implementation is able to handle all examples based on real applications in less than 100 seconds. Even though the size of the resulting formula is exponential in the number of agents in the universe, Aalta was still able to check the satisfiability of formulas consisting of thousands of LTL operators in reasonable time. As expected of a satisfiability solver, giving a counterexample for a formula is almost always faster than proving it unsatisfiable for formulas of comparable complexities.

The *Fixed-Arity* cases show that workflows handle an increasing number of relations with the same arity quite well. Here, adding another relation increases the size of the formula by a small factor, since the size of the needed universe stays the same - only the universally quantified encoding of the control flow graph grows. Increasing the necessary arity of the relations increases the minimum size of the universe - as shown by the *Increasing-Arity* cases. In case that all necessary agents are of the same sort, the formula grows exponentially, whereas it grows a lot slower in case that increasing the arity introduces a new sort. Since in those cases the size of the needed universe is exactly one agent per sort, the resulting LTL formula is even smaller than the FOLTL specification. The biggest factor in increasing the state space of the workflow, however, is the number of necessary causal agents as shown by the two variants of the *Causal-X* cases. Since every causal agent that we consider adds another copy of all of the agents needed to verify the workflow for only stubborn agents, adding the first causal agent doubles the minimum amount of agents in the universe. Since the size of the LTL formula is exponential in the number of agents, adding more causal agents causes the size of the resulting LTL formula to grow rapidly.

6 RELATED WORK

The closest work to ours is [16] where a similar workflow language is introduced. That language, however, does not provide control-flow constructs such as loops. Accordingly, a bounded model checking approach suffices to verify hyperproperties such as non-interference. In presence of loops, bounded model checking does no longer suffice for that purpose.

The workflow model that we consider is a type of a multi-agent system. Another type of multi-agent systems is represented by business processes. They are often described by BPMN diagrams [12] and formalized by Petri nets. A business process is a collection of activities, and a workflow thereof represents the flow of data items between activities. Activities are performed by users, who may need to synchronize on certain actions, but otherwise execute activities asynchronously. This is contrast to our formalism, where workflow steps are executed synchronously by a set of agents. Information flow in business processes has been considered, e.g., in [4]. There the MASK framework for possibilistic information flow security [26], and in particular a variant of the unwinding technique from [17], is used to prove that specifications satisfying particular constraints are safe. Up to our understanding, the approach is not easily amenable to automation.

There have recently been many efforts to verify concrete workflow systems, such as conference management systems [2, 21] or an

Table 2: Experiment Results

Name	# causal agents	Workflow size	Result	Universe size	FOLTL size	LTL size	Time (s)
Notebook	0	2	safe	(1,1)	266	240	0.18
Notebook	1	2	safe	(3,2)	309	993	2.92
Conference	0	5	safe	(2,1,1)	628	1089	2.50
Conference	1	5	unsafe	(4,2,2)	700	8771	91.86
Conference-acceptance	0	5	safe	(2,1)	628	1089	2.47
Conference-acceptance	1	5	unsafe	(4,2)	700	5187	45.63
Conference-linear	0	4	safe	(2,1)	469	698	0.75
Conference-linear	1	4	unsafe	(4,2,1)	541	4116	4.91
University	0	3	safe	(1,1)	305	202	0.01
University	2	3	unsafe	(4,3,3,2)	408	2727	1.28
Fixed-Arity-10	0	10	unsafe	(2)	1928	5299	0.89
Fixed-Arity-15	0	15	unsafe	(2)	3963	11114	3.09
Fixed-Arity-20	0	20	unsafe	(2)	6723	19054	16.85
Fixed-Arity-10-safe	0	10	safe	(2)	1924	5283	33.40
Fixed-Arity-15-safe	0	15	safe	(2)	3959	11098	158.83
Fixed-Arity-20-safe	0	20	safe	(2)	6719	19038	740.91
Increasing-Arity-2	0	2	safe	(2)	180	335	0.08
Increasing-Arity-3	0	3	safe	(3)	301	2206	6.20
Increasing-Arity-4	0	4	safe	(4)	451	21894	-
Sorted-Increasing-Arity-2	0	2	safe	(1,1)	180	163	0.03
Sorted-Increasing-Arity-3	0	3	safe	(1,1,1)	301	270	0.09
Sorted-Increasing-Arity-5	0	5	safe	(1,1,1,1,1)	630	559	0.40
Sorted-Increasing-Arity-10	0	10	safe	(1,...,1)	1960	1719	8.97
Causal-1	0	4	safe	(2)	654	1129	8.23
Causal-1	1	4	unsafe	(4)	747	2805	6.05
Causal-2	0	6	safe	(2)	778	1353	26.73
Causal-2	2	6	unsafe	(6)	965	6338	195.31
Sorted-Causal-2	2	3	unsafe	(4,3,1)	378	1184	1.47
Sorted-Causal-3	3	4	unsafe	(5,4,4,1)	598	3169	1.96
Sorted-Causal-5	5	5	unsafe	(7,6,6,6,6,1)	1197	14510	17.05

eHealth system [7], or a social media platform [5]. For instance, the CoCon conference management system [21] is implemented and checked in the interactive theorem prover Isabelle. Its security model uses a specialized non-interference notion (based on noninducibility [30]), which is motivated by the need for fine-grained declassification conditions. In our case, this need is satisfied by the use of FOLTL, which allows for specifying fine-grained declassification conditions both in the "what" and in the "when" directions [28]. ConfiChair [2] is a cryptographic-based model of a cloud-based conference management system for which the strong secrecy (also a hyperproperty) of paper contents and reviews is automatically checked with the ProVerif tool. In contrast to these works, which focus on the verification of one specific system, we propose a modeling language for workflow, together with a verification approach.

Another attempt to verify parametric systems via a formalization in first-order logic is the CSDN language [3]. That language has been proposed for describing and verifying the semantics of controllers in software-defined networks (SDNs). With our workflow language, it shares that the semantics is specified in terms of relations. A CSDN program consists of a sequence of controller rules, each guarded by an event pattern. When the event pattern fires, the corresponding command is executed. Commands are expressed in a simple imperative language, which allows to query and update relations. In contrast, the basic step of our workflow language is the **forall** block, which consists of a sequence of guarded updates to relations. This sequence is executed *in parallel* for each instantiation of the block variables. Accordingly, the semantics in [3] and the one in the present paper are orthogonal, and cannot easily simulate one another. Furthermore, in contrast to [3], we are not only interested in plain invariants, but temporal non-interference properties expressed by HyperFOLTL.

Expressing trace properties with sorted FOLTL has been initiated already in the work of Manna and Pnueli [25] and logic-based approaches are now standard in the verification of such properties. Logic-based approaches for non-trace properties are less common and include the ones based on epistemic temporal logics [14], SecLTL [13], and in particular HyperLTL [9], the logic whose first-order extension we use in this paper.

7 CONCLUSION

We have provided an extension to the workflow language from [16] with non-deterministic control-flow structures. We have encoded

the semantics of these workflows as well as complex non-interference properties into sorted FOLTL and identified a fragment of sorted FOLTL where satisfiability is decidable. From that, we concluded that non-interference is decidable for non-omitting workflows and a fixed number of causal agents. These methods are strong enough to automatically construct attacks to the example property.

We also explored in how far our decidability result can be further generalized. We found, however, that dropping either the restriction on workflows or the bound on the number of causal agents results in undecidability.

We have implemented the tool NIWO which automatically verifies sorted non-interference properties for non-omitting workflows. We evaluated our implementation on workflows inspired by a conference management system. Nonetheless, we would like to see specifications of larger workflows in order to better understand the potentials and limitations of our methods.

A practical verification system for arbitrary workflows in our language requires to deal with the satisfiability problem of general (sorted) FOLTL formulas. Clearly, FOLTL is a fragment of FOL— using one unary function symbol. It remains for future work to explore in how far current automated theorem provers such as Spass [32] or Z3 [11], or model-finders such as Alloy [19] or its temporal extension Electrum [24] are able to deal with the non-interference formulas for workflows; or what extra proving technology is required.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. This work was partially supported by the German Research Foundation (DFG) under the project "SpAGAT" (grant no. FI 936/2-1) in the priority program "Reliably Secure Software Systems - RS3", in the doctorate program "Program and Model Analysis - PUMA" (no. 1480), and as part of the Collaborative Research Center "Methods and Tools for Understanding and Controlling Privacy" (SFB 1223).

REFERENCES

[1] Aharon Abadi, Alexander Rabinovich, and Mooly Sagiv. 2010. Decidable fragments of many-sorted logic. *Journal of Symbolic Computation* 45, 2 (2010), 153–172.

[2] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. 2012. Privacy Supporting Cloud Computing: ConfiChair, a Case Study. In *Proc. POST 2012*. Springer Verlag, 89–108.

[3] Thomas Ball, Nikolaj Bjørner, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Mooly Sagiv, Michael Schapira, and Asaf Valadarsky. 2014. VeriCon: towards verifying controller programs in software-defined networks. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2014)*. ACM, 282–293.

[4] Thomas Baeureiß and Dieter Hutter. 2014. Information flow control for workflow management systems. *it - Information Technology* 56, 6 (2014), 294–299.

[5] Thomas Baeureiß, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. 2017. CoSMeDis: A Distributed Social Media Platform with Formally Verified Confidentiality Guarantees. (2017). to appear in Security and Privacy 2017.

[6] Robert Berger. 1966. *The undecidability of the domino problem*. Number 66. American Mathematical Soc.

[7] C. Bhardwaj and S. Prasad. 2015. Parametric information flow control in ehealth. In *Proceedings HealthCom 2015*. 102–107. https://doi.org/10.1109/HealthCom.2015.7454481

[8] Egon Börger, Erich Grädel, and Yuri Gurevich. 1997. *The Classical Decision Problem*. Springer.

[9] Michael R Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K Micinski, Markus N Rabe, and César Sánchez. 2014. Temporal logics for hyperproperties. In *International Conference on Principles of Security and Trust*. Springer, 265–284.

[10] Michael R. Clarkson and Fred B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[11] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[12] Remco M Dijkman, Marlon Dumas, and Chun Ouyang. 2008. Semantics and analysis of business process models in BPMN. *Information and Software technology* 50, 12 (2008), 1281–1294.

[13] R. Dimitrova, B. Finkbeiner, M. Kovács, M. N. Rabe, and H. Seidl. 2012. Model Checking Information Flow in Reactive Systems. In *Proc. VMCAI'12*. 169–185.

[14] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. 1995. *Reasoning About Knowledge*. MIT Press.

[15] Bernd Finkbeiner and Christopher Hahn. 2016. Deciding Hyperproperties. In *27th International Conference on Concurrency Theory (CONCUR 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Josée Desharnais and Radha Jagadeesan (Eds.), Vol. 59. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 13:1–13:14.

[16] Bernd Finkbeiner, Helmut Seidl, and Christian Müller. 2016. Specifying and Verifying Secrecy in Workflows with Arbitrarily Many Agents. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA 2016) (Lecture Notes in Computer Science)*, Vol. 9938. 157–173.

[17] J. A. Goguen and J. Meseguer. 1982. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*. 11–20.

[18] Ian M. Hodkinson, Frank Wolter, and Michael Zakharyaschev. 2000. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Logic* 106, 1-3 (2000), 85–134.

[19] Daniel Jackson. 2012. *Software Abstractions: logic, language, and analysis*. MIT press.

[20] Emmanuel Jeandel. 2010. The periodic domino problem revisited. *Theoretical Computer Science* 411, 44-46 (2010), 4010–4016.

[21] Sudeep Kanav, Peter Lammich, and Andrei Popescu. 2014. A Conference Management System with Verified Document Confidentiality. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV 2014)*. Springer Verlag, 167–183.

[22] Denis Kuperberg, Julien Brunel, and David Chemouil. 2016. On Finite Domains in First-Order Linear Temporal Logic. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 211–226.

[23] Jianwen Li, Yinbo Yao, Geguang Pu, Lijun Zhang, and Jifeng He. 2014. Aalta: An LTL Satisfiability Checker over Infinite/Finite Traces. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, 731–734.

[24] Nuno Macedo, Julien Brunel, David Chemouil, Alcino Cunha, and Denis Kuperberg. 2016. Lightweight specification and analysis of dynamic systems with rich configurations. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. ACM, 373–383.

[25] Zohar Manna and Amir Pnueli. 1981. Verification of Concurrent Programs: The Temporal Framework. In *The Correctness Problem in Computer Science*, Robert S. Boyer and J Strother Moore (Eds.). Academic Press, London, 215–273.

[26] Heiko Mantel. 2000. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society, 185–199.

[27] Timothy Nelson, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2012. Toward a More Complete Alloy. In *Proceedings of the 3rd International Conference on Abstract State Machines, Alloy, B, VDM, and Z (ABZ 2012) (Lecture Notes in Computer Science)*, Vol. 7316. Springer, 136–149.

[28] A. Sabelfeld and D. Sands. 2005. Dimensions and Principles of Declassification. In *Proceedings CSFW'05*. IEEE Computer Society, 255–269.

[29] A. P. Sistla and E. M. Clarke. 1985. The Complexity of Propositional Linear Temporal Logics. *J. ACM* 32, 3 (July 1985), 733–749.

[30] David Sutherland. 1986. A model of information. In *Proc. 9th National Computer Security Conference*. DTIC Document, 175–183.

[31] Hao Wang. 1990. Dominoes and the AEA case of the decision problem. In *Computation, Logic, Philosophy*. Springer, 218–245.

[32] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. 2009. SPASS Version 3.5. In *International Conference on Automated Deduction*. Springer, 140–145.

[33] S. Zdancewic and A. C. Myers. 2003. Observational Determinism for Concurrent Program Security. In *Proceedings of CSFW'03*.