Inductive Invariants for Noninterference in Multi-agent Workflows

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

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

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

***Abstract*—Our goal is to certify absence of information leaks in multi-agent workflows, such as conference management systems like EASYCHAIR. These workflows can be executed by any number of agents some of which may form coalitions against the system. Therefore, checking noninterference is a challenging problem. Our paper offers two main contributions: First, a technique is provided to translate noninterference (in presence of various agent capabilities and declassification conditions) into universally quantified invariants of an instrumented new workflow program. Second, general techniques are developed for checking and inferring universally quantified inductive invariants for workflow programs. In particular, a large class of workflows is identified where inductiveness of invariants is decidable, as well as a smaller, still useful class of workflows where the weakest inductive universal invariant implying the desired invariant, is effectively computable. The new algorithms are implemented and applied to certify noninterference for workflows arising from conference management systems.**

***Index Terms*—multi-agent workflows; non-interference; abstract interpretation; inductive invariants**

I. INTRODUCTION

A multi-agent system, e.g., for the management of a conference, may be used by any number of agents, while at the same time posing non-trivial restrictions to the flow of information. A challenging problem therefore is to certify noninterference — independently of the number of participating agents. Workflows, as presented in [11], [10], describe a language for multi-agent systems in which agent interactions are structured in stages and recorded by relations, with each stage expressed as a block of guarded updates to relations. For instance, one stage of a conference management workflow could be specified by the following workflow block:

$$\textbf{forall } x, y, p, r \textbf{ may}. \quad \mathit{Assign}(x,p) \wedge \mathit{Review}(y,p,r) \rightarrow \mathit{Read} \mathrel{+}= (x,p,r)$$

which states that each PC member x may decide to read (or not to read) the review r of the paper p, given that x is assigned to p and some PC member y has provided the review for p.

In such a system, a security breach occurs if an agent obtains information that she is not entitled to know, like a PC member obtaining reviews on papers she is in conflict with. The absence of information leaks is best expressed as a noninterference-like property — a property on two traces, roughly stating that from the point of view of any single agent, there is no observable difference between two executions of the workflow that differ in their secret information, as long as the behavior of the other participating agents is *reasonable* (otherwise the property specification allows for spurious attacks). In [11], it is argued that *reasonable* behavior means that the decisions or choices of an agent can only reveal information the agent has actually observed. Such agents are called *causal*. This causality assumption, while preventing false negatives, still allows agents to form coalitions. Indeed, knowledge of confidential information can be transmitted by agents via the side channel of adapting their observable behaviors. A stronger assumption on the agent behavior, also introduced in [11], is that she acts *stubbornly*: stubborn agents show the same behavior (in the two executions) independently of their knowledge. They thus can propagate information about confidential data to other agents only directly.

Besides the specification of the abilities of the participating agents, realistic noninterference properties for workflows also require a specification of *declassification conditions*, as sensitive information will be revealed to some agents and considered secret for others.

To formalize such complex security properties, which encompass the assumptions on the agent behavior and the declassification conditions, both [11] and [10] use a first-order extension of HyperLTL [6]. In [10], the semantics of workflows as well as noninterference properties are translated into sorted FOLTL. Quite general forms of noninterference could be proven decidable for the restricted class of *non-omitting* workflows by means of a translation into a decidable fragment of sorted FOLTL.

Here, a workflow is *non-omitting* if all occurring guards are quantifier-free, and every modification of a predicate always contains *all* involved agents explicitly. Many practically useful workflows, though, *are* omitting, like the example workflow block shown above, where y does not occur in the tuple (x, p, r) to be inserted into the relation for Read. In general, statements which use auxiliary variables for expressing guards, naturally introduce omission. A simple and common example is a statement that expresses the transitive closure R of some relation E:

$$\textbf{forall } x, y, z.\ E(x,y) \wedge R(y,z) \rightarrow R \mathrel{+}= (x,z)$$

We note here that this block can be equivalently rewritten as

$$\textbf{forall } x, z.\ \exists y.\ E(x,y) \wedge R(y,z) \rightarrow R \mathrel{+}= (x,z).$$

However, the resulting workflow is still omitting as the guard is not quantifier-free.

Therefore, the results from [10], while encouraging, can only be a first step. In [10], however, it has also been shown that already for non-omitting workflows and an unbounded number of *causal* agents, noninterference with declassification is undecidable, while for omitting workflows undecidability already occurs for the simplest agent model, namely, for *stubborn* agents. In that sense, the results of [10] are tight. In this article, we nonetheless aim at lifting the non-omission restriction, and thus provide practical methods for analyzing general workflows with unboundedly many agents, causal or stubborn. Due to the undecidability results of [10], however, any verification approach is necessarily incomplete and/or introduces further restrictions. We therefore concentrate on noninterference (instead of temporal hyper-properties) with declassification depending on the current state only, and consider proofs using *universally* quantified formulas only.

The key observation behind our approach is that, for pairs of execution traces complying with the assumptions on agent behavior and with the declassification conditions, the specification of noninterference boils down to checking that all observations of a particular agent coincide on the two traces. That is, indistinguishability of such traces can be cast as a *universal invariant*, i.e., a mapping from program points of the workflow to universally quantified formulas. Therefore, in contrast to [11], [10], we propose in this paper not to include the assumptions on agent behavior and declassification into the specification of noninterference. Instead, we transform the given workflow w so that the resulting workflow now tracks *pairs* of execution traces of w that comply both with the assumptions on agents and declassification. Proving that the workflow satisfies an invariant can then be done by providing a strengthening of the invariant which is *inductive*. We find that it is decidable whether an assignment of formulas to the nodes of a control flow graph of the resulting workflow is inductive or not — whenever all formulas in question are universal, and the weakest precondition calculation for each control flow edge only introduces non-nested occurrences of quantifiers. This provides us with a means for *proving* noninterference with declassification for omitting workflows.

Checking an assignment for being inductive, is one thing; *inferring* an inductive invariant which is sufficiently strong for proving the given assignment, is another. The latter can be cast as finding a solution to a constraint system for weakest preconditions of the invariant to be verified [7]. Even if only universal properties are of concern, the underlying lattice for solving that constraint system has infinitely descending chains — implying that fixpoint iteration may not terminate. Based on an abstraction technique for formulas possibly containing existential quantification, and second-order quantifier elimination, we identify non-trivial cases where the fixpoint iteration can be proven to terminate and then return the *weakest* inductive invariant, i.e., the inductive invariant making the least assumptions on the program states. An implementation of our method demonstrates that the approach is able to deal with more general workflows while still being faster than the implementation provided for [10] — whenever that is applicable.

The paper is organized as follows. After a brief recap of the workflow model and a formalization of noninterference in Section II, we present our transformation to encode agent model as well as declassification into an ordinary workflow in Section III. By this transformation, checking of noninterference can be reduced to checking a universal invariant. Section IV therefore considers universal invariants and proves that for these, inductiveness is decidable. Section V presents a method for *inferring* inductive invariants. This method relies on abstract interpretation techniques for first-order formulas and proves optimality under mild assumptions — given that fixpoint iteration for an appropriate constraint system terminates. In detail, termination is treated in Section VI where a termination proof is presented for a natural syntactic restriction of workflows. In Section VII the application of the given techniques for proving noninterference is wrapped up. In particular, it is shown that checking noninterference for a bounded number of causal agents is not more difficult than checking of universal invariants of workflows. Section VIII then presents an implementation and experimental results.

II. Noninterference for Workflows

A. Workflows

We fix a finite set $\mathcal{R}$ of relation symbols. We also fix a set $\mathcal{V}$ of first-order variables and a subset $\mathcal{X} \subseteq \mathcal{V}$ of variables occurring *freely* in the specification. These can be considered as constants from the universe to be interpreted. The set $\mathcal{R}$ is the disjoint union of the subsets $\mathcal{R}_{wf}$, $\mathcal{R}_o$, and $\mathcal{R}_c$. Relations in $\mathcal{R}_{wf}$ may be updated by the workflow at each step. Relations in $\mathcal{R}_o$ are provided by the environment — they can only be queried and are therefore called *oracles*. They may contain confidential data and come with a *declassification condition* (as detailed later in this section). The relations in $\mathcal{R}_c$, on the other hand are not explicitly mentioned in the workflow. These represent the decisions taken by agents participating in the workflow and are therefore called *choices*. We follow the convention that agent capabilities are specified via the first components of tuples. Concretely, an agent a can observe all tuples of workflow relations that mention her in the first component (i.e. all tuples of the form $(a, \bar{b})$ that satisfy some $R \in \mathcal{R}_{wf}$). Also, an agent chooses all tuples of the choice predicates $C \in \mathcal{R}_c$ that mention her in the first component.

A workflow w is defined by the grammar given in Fig. 1, where x_i, y_i range over variables in $\mathcal{V}$, R ranges over predicate symbols in $\mathcal{R}_{wf}$, and φ ranges over first-order formulas, possibly containing equality, with relation symbols in $\mathcal{R}_{wf} \uplus \mathcal{R}_o$. The basic constructs are *statements*, which represent the simultaneous updates of relations. As updates we only consider guarded additions or deletions of individual tuples, simultaneously executed for all values of the occurring variables. Sequences of statements are grouped into *blocks*. Their **forall** prefix lists the variables over which the simultaneous updates range. By convention, the first variable in the list refers to the agent executing the block. The key word **may** indicates that the agent in the first component of the tuple may *choose* to participate in the execution of the block or desist. Technically, this means for the semantics of these blocks, that the literal $C(\bar{x})$ is added to each guard occurring in the block ($\bar{x}$ being the sequence of variables listed in the **forall** prefix, C a choice predicate of appropriate arity). Finally on the top-level, blocks are organized by sequencing, alternatives, and loops, where a particular execution is meant to be determined by external decisions (in case of a conference management system, e.g., by deadlines or the PC chair).

For the syntax and semantics of the first-order formulas serving as guards, we refer to standard textbooks such as [9]. For a statement $\varphi \to R \pm\!= \bar{y}$, we require that $|\bar{y}|$ matches R's arity. The formula φ as well as the sequence $\bar{y} = (y_1, \ldots, y_l)$ may contain variables not in $\bar{x}$, where $\bar{x} = (x_1, \ldots, x_k)$ is the sequence of variables appearing in the **forall** construct of the surrounding block. These variables not in $\bar{x}$ are assumed to be in $\mathcal{X}$, i.e., global constants of the workflow.

In the following, we introduce further restrictions not imposed in [10] — these are mild restrictions that do not exclude any of our current examples. We assume 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 $\to$). We also assume that oracles are queried only in non-**may** blocks, and in each block that queries an oracle, the same oracle is queried. Furthermore each guard querying an oracle O is of the form $\varphi' \wedge O\bar{x}$ for some formula φ', where the sequence of variables $\bar{x}$ coincides with the block's quantified variables (i.e. the block is of the form **forall** $\bar{x}.\,\mathit{stmts}$). This restriction is mainly used to simplify arguments later in the paper — in Appendix A. we provide a transformation that can be used to transform any given workflow into one that fits these constraints for oracle usage.

$$
\begin{array}{lcl}
w & ::= & \epsilon \mid \mathit{block}\ w' \mid \textbf{loop}\ \{w'\}\ w'' \mid \\
 & & \textbf{choose}\ \{w_1\}\ \textbf{or}\ \{w_2\}\ w' \\
\mathit{block} & ::= & \textbf{forall}\ x_1, \ldots, x_k.\ \mathit{stmts} \\
 & & \mid \textbf{forall}\ x_1, \ldots, x_k\ \textbf{may}.\ \mathit{stmts} \\
\mathit{stmts} & ::= & \epsilon \mid \mathit{stmt}; \mathit{stmts} \\
\mathit{stmt} & ::= & \varphi \to R \mathrel{+}= (y_1, \ldots, y_l) \\
 & & \mid \varphi \to R \mathrel{-}= (y_1, \ldots, y_l)
\end{array}
$$

Figure 1. Definition of a workflow w

For completeness, we recall the definition of non-omitting workflows, a critical assumption imposed in [10], but not here. A workflow is called *non-omitting* iff for each block of the form **forall** $\bar{x}$ [**may**] stmts, and for each of its statements $\varphi \to R \pm\!= \bar{y}$ in a block, we have that φ is quantifier-free and each variable in $\bar{x}$ also appears in $\bar{y}$.

Example 1. *The workflow*

$$\textbf{forall}\ x_1, x_2.\ \textbf{may}\ \mathit{true} \to R \mathrel{+}= (x_2, c, x_1)$$

is non-omitting (when c is a constant), whereas the workflow

$$\textbf{forall}\ x_1, x_2.\ \textbf{may}\ R(x_2) \to S \mathrel{-}= (x_1, c)$$

is omitting, since x_2 is not mentioned in the tuple (x_1, c) that is removed from S.

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

Figure 2. EASYCHAIR-like workflow.

Example 2. *As a running example, we use a similar workflow as in [10], given in Fig. 2. The workflow 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.*

We emphasize that the corresponding workflow from [10] is non-omitting, while this workflow is omitting, because in block (b_4) the variable y does not occur in the symbolic tuple (x, p, r), which represents the concrete tuples that are added to the Read relation.

B. Semantics

We give the semantics of workflows in terms of (symbolic) transition systems based on control flow graphs (CFGs). The CFG for the workflow in Fig. 2 is given in Fig. 3. The control flow graph of a workflow consists of:

- a finite set V of *control points* or *nodes* including $u_0, u_t \in V$ as initial and terminal node, respectively;
- a finite set of edges E where each edge in E is of the form (u, β, v) where $u, v \in V$ are the start and end nodes of the edge, and β is a sequence of blocks (usually just one). We remark that β can also be an *empty* sequence of blocks, in which case we also denote it by nop.

We assume that each node has at least one outgoing edge. In particular, the terminal node has (u_t, nop, u_t) as single

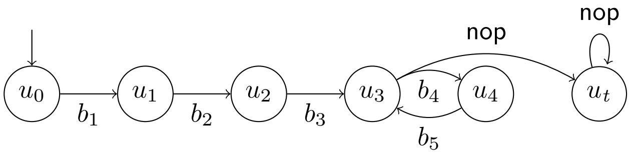

Figure 3. CFG of the workflow in Example 2.

outgoing edge. This convention 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 where the attained *states* are changed according to the labels of the traversed edges. In the following, we view a workflow w as given by such a control flow graph.

Consider a fixed universe U and a valuation $\rho : \mathcal{X} \to U$ of the constants. Assume that the set $\mathcal{R}_c$ contains a relation symbol C_b for every **may** block b of the form **forall** $\bar{x}$ **may**. *stmts*; C_b's arity is $|\bar{x}|$. Then a *state* s is a first-order structure which interprets the symbols in $\mathcal{R}$ as relations (of right arities) over U. The state s is *initial*, iff all predicates in $\mathcal{R}_{wf}$ are empty. This means that $s, \rho \models \varphi_0$ where φ_0 is the conjunction of all formulas $\forall \bar{z}. \neg R\bar{z}$, for $R \in \mathcal{R}_{wf}$. The notation $s, \nu \models \psi$ for a structure s over U, a valuation ν, and a first-order formula ψ will be used throughout this paper to denote that formula ψ holds in the structure s where each free variable in ψ is interpreted according to ν. We often write $R\bar{z}$ for $R(\bar{z})$.

Now assume that the execution of the workflow has reached some control point u in state s, and an edge (u, β, u'). Let next_β denote the relation consisting of all pairs (s, s') so that s' can be reached from s by successively executing the blocks in β. In order to define this relation, we first consider a single block b. For b, we introduce the substitution θ_b of atomic predicates in $\mathcal{R}_{wf}$ which is defined as follows. Assume first that b is a **may** block of the form **forall** $\bar{x}$ **may**. *stmts*. Then, for each $R \in \mathcal{R}_{wf}$, we define

$$\theta_b(R\bar{z}) := \begin{cases} R\bar{z} & \text{if (a)} \\ R\bar{z} \vee \exists \bar{x}.\, \varphi \wedge C_b\bar{x} \wedge (\bar{z} \doteq \bar{y}) & \text{if (b)} \\ R\bar{z} \wedge \neg\big(\exists \bar{x}.\, \varphi \wedge C_b\bar{x} \wedge (\bar{z} \doteq \bar{y})\big) & \text{if (c)} \end{cases}$$

where the conditions (a-c) are: (a) R is not updated, (b) $\bar{y}$ is added to R, (c) $\bar{y}$ is deleted from R; and it is assumed that $\bar{x} \cap \bar{z} = \emptyset$ and that the statement in *stmts* that updates R (if any) has the form $\varphi \to R \mathrel{\pm}= \bar{y}$. The definition of θ_b when b is a non-**may** block is similar, except that the $C_b\bar{x}$ conjuncts are omitted.

For a sequence $\beta = b_1 \ldots b_h$ consisting of the blocks $b_1, \ldots, b_h$, $\theta_\beta = \theta_{b_1} \circ \ldots \circ \theta_{b_h}$ is the composition of the substitutions θ_{b_i} for the blocks b_i in β. (θ_β is the identity when β is empty.) Then $(s, s') \in \mathsf{next}_\beta$ iff each relation $R \in \mathcal{R}_{wf}$ in s' consists of all tuples $\bar{a}$ so that $s, \rho[\bar{z} \mapsto \bar{a}] \models \theta_\beta(R\bar{z})$, where $\rho[\bar{z} \mapsto \bar{a}]$ is the valuation which extends ρ by assigning to each variable in the sequence $\bar{z}$ the corresponding value in the sequence $\bar{a}$. The interpretations of the predicates of $\mathcal{R}_o \cup \mathcal{R}_c$ in s and s' are unrelated.

As variables in $\bar{y}$ either occur in $\bar{x}$ or in $\mathcal{X}$, we remark that the formulas $\theta_b(R\bar{z})$ can be simplified by removing the equalities $z_i \doteq y_i$, and replacing z_i with the constant a if $y_i = a \in \mathcal{X}$ and replacing x_j with z_i and removing the existential quantification on x_j if $y_i = x_j$ for some j.

Example 3. *For the block* b

$$\textbf{forall } x_1, x_2, x_3 \textbf{ may}.\ P(x_1, x_2) \to R \mathrel{+}= (x_3, a, x_1),$$

we have

$$\begin{aligned} &\theta_b(R(z_1, z_2, c)) \\ = {} & R(z_1, z_2, c) \vee \exists x_1, x_2, x_3.\, P(x_1, x_2) \wedge \\ & C_b(x_1, x_2, x_3) \wedge z_1 \doteq x_3 \wedge z_2 \doteq a \wedge c \doteq x_1 \\ \equiv {} & R(z_1, a, c) \vee \exists x_2.\, P(c, x_2) \wedge C_b(c, x_2, z_1) \end{aligned}$$

For *non-omitting* workflows, the existential quantification disappears completely by using the given simplification.

A *trace* of the workflow (now given by its CFG) is an infinite sequence

$$\tau = (u_0, s_0, \beta_0), (u_1, s_1, \beta_1), \ldots$$

where s_0 is an initial state, and for each $i \geq 0$, (u_i, β_i, u_{i+1}) is a control flow edge where $(s_i, s_{i+1}) \in \mathsf{next}_{\beta_i}$. The control flow path $(u_0, \beta_0), (u_1, \beta_1), \ldots$ and the sequence of states $s_0, s_1, \ldots$ of the trace τ is denoted by $\pi(\tau)$ and $\sigma(\tau)$, respectively. We use FOLTL to describe trace properties of workflows. We refer, e.g., to [10] for the definition of FOLTL. For a sequence of structures $\bar{s} = s_0 s_1 \ldots$ over the same universe U, a valuation ν over U, and an FOLTL formula ψ, we use the notation $\bar{s}, \nu \models \psi$ if ψ holds for $\bar{s}$ when the free variables in ψ are interpreted by means of ν. For ψ with free variables in $\mathcal{X}$, we write $w \models \psi$ if $\sigma(\tau), \rho \models \psi$, for each trace τ of w.

C. Noninterference

Noninterference is best formulated as a 2-*hyperproperty* [6], that is, a property of pairs of traces. In our application, the sequence of edges traversed by an execution of the workflow is determined *externally*, i.e., independent of any oracle or choice predicate. For instance in case of a conference management system, it is up to the PC chair to decide when a particular stage is complete and which next stage to execute. This means that we are only interested in 2-hyperproperties where the considered two traces follow the *same* control flow path, but may differ in the sequences of attained states. This restriction has also been imposed in [11], [10].

Consider traces $\tau = (u_0, s_0, \beta_0), (u_1, s_1, \beta_1), \ldots$ and $\tau' = (u_0, s'_0, \beta_0), (u_1, s'_1, \beta_1), \ldots$ of the workflow w which agree in their control flow paths, i.e., $\pi(\tau) = \pi(\tau')$. Then we can combine these into the sequence $\tau \otimes \tau' = (u_0, \langle s_0, s'_0 \rangle, \beta_0), (u_1, \langle s_1, s'_1 \rangle, \beta_1), \ldots$. In order to reason about the pairs of states attained in $\tau \otimes \tau'$, we introduce a copy $\mathcal{R}' = \{R' \mid R \in \mathcal{R}\}$ of the predicates in $\mathcal{R}$ and assume that the states s'_i are expressed by means of the predicates in $\mathcal{R}'$. Thus, we can combine each pair $\langle s_i, s'_i \rangle$ of first-order structures into a single structure $s_i \otimes s'_i$ over $\mathcal{R} \cup \mathcal{R}'$. The sequence $s_0 \otimes s'_0, s_1 \otimes s'_1, \ldots$ is also referred to as $\sigma(\tau \otimes \tau')$. Given an FOLTL formula ψ over $\mathcal{R} \cup \mathcal{R}'$, we write $w \models_2 \psi$ if $\sigma(\tau \otimes \tau'), \rho \models \psi$ for each pair of traces τ, τ' with

$\pi(\tau) = \pi(\tau')$. We thus use FOLTL to specify trace properties, as well as 2-hyperproperties such as noninterference.

Following [11], [10], noninterference is expressed from the point of view of a single (but arbitrary) agent, and the notions of high/low inputs/outputs from the standard definition of noninterference [13] are interpreted with respect to this agent. Furthermore, the property is parameterized by an assumption, called *agent model*, on the behavior of agents, and by a *declassification condition*, which specifies when and what information can be legitimately exposed. Therefore, *Noninterference with Declassification and Agent model* (NDA) is expressed by the FOLTL formula

$$\mathit{agent\_model} \to \forall a.\, \big(\mathsf{G}\, \mathit{eqOracles}(a)\big) \to \big(\mathsf{G}\, \mathit{eqObs}(a)\big)$$

where G is the LTL "globally" operator. The property states that for any two traces, under a given agent model, for every agent a, the noninterference property holds iff agent a is never able to observe a difference between two executions that differ only in the (non-declassified) inputs from the oracles:

$$\mathit{eqObs}(a) := \textstyle\bigwedge_{R\in\mathcal{R}_{\mathit{wf}}} \big(\forall \bar{z}.\, Ra\bar{z} \leftrightarrow R'a\bar{z}\big)$$
$$\mathit{eqOracles}(a) := \textstyle\bigwedge_{O\in\mathcal{R}_o} \forall \bar{z}.\, \big(\mathit{declass}_O(a,\bar{z}) \to (O\bar{z} \leftrightarrow O'\bar{z})\big)$$

For each oracle predicate O, the formula $\mathit{declass}_O(a,\bar{z})$ using predicates from $\mathcal{R}$ (not from $\mathcal{R}'$) encodes a declassification condition that specifies which tuples from O can be made visible to a without causing a security breach. For our running example, we use $\mathit{declass}_O(a,x,p,r) := \neg \mathit{Conf}(a,p)$.

D. Agent Model

For any agent, we consider two kinds of possible behavior. One agent either *stubbornly* makes the same choices, independently of its observations; or its choices may depend on previous observations, i.e., it acts *causally*. These two behaviors are captured by the following formulas, respectively.

$$\mathit{stubborn}(a) := \mathsf{G}\, \mathit{eqChoices}(a)$$
$$\mathit{causal}(a) := \mathit{eqChoices}(a)\, \mathsf{W}\, \neg \mathit{eqObs}(a)$$

where

$$\mathit{eqChoices}(a) := \bigwedge_{C\in\mathcal{R}_c} \big(\forall \bar{z}.\, Ca\bar{z} \leftrightarrow C'a\bar{z}\big)$$

and W denotes the "weak until" LTL operator. Note that any stubborn agent is also a causal one, thus the causality assumption allows for more behaviors. Therefore, the most general agent model is when each agent is causal, while the most restrictive model is when each agent is stubborn. In [10], it has been shown that checking NDA for an unbounded number of causal agents is undecidable.

We thus assume that the *agent_model* formula from the formalization of NDA can be instantiated with one of the following formulas:

$$\mathit{agent\_model}^{(c,k)} := \exists a_1,\ldots,a_k.\, \big(\textstyle\bigwedge_{i=1}^{k} \mathit{causal}(a_i)\big) \wedge \big(\forall a.\, (\textstyle\bigwedge_{i=1}^{k} a \neq a_i) \to \mathit{stubborn}(a)\big)$$
$$\mathit{agent\_model}^{(c)} := \forall a.\mathit{causal}(a)$$

where $k \geq 0$. Note that $\mathit{agent\_model}^{(c,0)} \equiv \forall a.\mathit{stubborn}(a)$. We denote this formula by $\mathit{agent\_model}^{(s)}$.

III. Encoding Agent Models and Declassification

The key issue of NDA is to verify that only the same observations are possible — on every trace satisfying the given sanity requirements. Instead of formalizing these requirements by means of temporal logic formulas, we introduce a fresh agent variable a and transform the workflow w in such a way that the resulting workflow, for any agent value of a, simultaneously computes the states attained at two traces sharing the same control flow path, and additionally enforces the sanity requirements for a on these states. NDA then amounts to verifying a universal invariant of the transformed workflow. Next, we instantiate this idea for causal agents.

Assume we are given a workflow w (via its CFG), and a variable a, corresponding to the agent of concern for noninterference. We then construct a new workflow $\mathcal{T}_a^{(c)} w$ as follows. Let $\mathcal{R}'$ denote the set of primed predicates R' corresponding to the predicates R used by w. For a first-order formula φ with predicates from $\mathcal{R}$, let $[\varphi]'$ denote the formula obtained from φ by replacing each predicate $R \in \mathcal{R}$ with the corresponding predicate R' in $\mathcal{R}'$. Then each edge (u, β, v) of w gives rise to one edge $(u, \mathcal{T}_a^{(c)}\beta, v)$ in $\mathcal{T}_a^{(c)} w$ where the transformation $\mathcal{T}_a^{(c)}$ on sequences of blocks is the concatenation of the transformations of the individual blocks b in β. For a given block b, the transformation $\mathcal{T}_a^{(c)} b$ is the sequence $b_1 b_2 \beta'$, where b_1, b_2 represent transformations of b that handle the updates to workflow relations, while the sequence β' of blocks handles the causality assumption. To define b_1, b_2, β', we make a case distinction depending on the type of the block b, i.e. whether or not it is a **may**-block, and if not, whether or not it queries an oracle. If b is non-**may** block, then β' is the empty sequence.

Parallel Updates. For a non-**may** block b that does not contain a query to an oracle, the transformed workflow will simply contain updates to both copies of the relation. In this case, b_1 is equal to b and b_2 is obtained from b by replacing every update $\varphi \to R \mathrel{+}= \bar{z}$ by the update $[\varphi]' \to R' \mathrel{+}= \bar{z}$, and likewise, every update $\varphi \to R \mathrel{-}= \bar{z}$ by the update $[\varphi]' \to R' \mathrel{-}= \bar{z}$.

Declassification. For non-**may** blocks that query an oracle, the transformed workflow needs to simulate the semantics of the declassification conditions. Thus, the blocks b_1, b_2 are **may**-blocks, where the predicates C_{b_1}, C_{b_2} serve as the possibly distinct versions of the oracle on the two simulated traces. Assume b is of the form **forall** $\bar{x}.\mathit{stmts}$ and contains a query to the oracle O. Then block b_1 equals **forall** $\bar{x}$ **may**. stmts_1 where the sequence stmts_1 is obtained from *stmts* by adding two new updates for every update $\varphi \wedge O\bar{x} \to R \mathrel{+}= \bar{y}$ in *stmts*:

$$\varphi \to R \mathrel{+}= \bar{y}$$
$$[\varphi]' \wedge \mathit{declass}_O(a,\bar{x}) \to R' \mathrel{+}= \bar{y}$$

Moreover, b_2 equals **forall** $\bar{x}$ **may**. stmts_2 where the sequence stmts_2 is obtained from *stmts* by adding one new update for every update $\varphi \wedge O\bar{x} \to R \mathrel{+}= \bar{y}$ in *stmts*:

$$[\varphi]' \wedge \neg \mathit{declass}_O(a,\bar{x}) \to R' \mathrel{+}= \bar{y}$$

This ensures that if declassification holds, then the updates to R and R' use the same predicate C_{b_1}, and otherwise they use distinct ones (i.e. C_{b_1} and C_{b_2}). In both b_1 and b_2, updates that query the oracle, but remove tuples, are treated analogously.

Causality. The remaining case is when b is of the form **forall** $x, \bar{y}$ **may**. *stmts*. As agents are causal, they may make different choices when executing such blocks, provided that they have already observed a difference in the considered traces. (In non-**may** blocks, no choices are made.) Therefore, in order to capture causality, we introduce a new unary predicate $\mathit{Informed}(x)$, which is used to record all agents x that have already made observations depending on secret information — so their choices can diverge. Causality then states that the corresponding choice relations must be the equal for x whenever $\neg\mathit{Informed}(x)$ holds. We will use the predicates C_{b_1}, C_{b_2} to encode this conditional equivalence. In case the choice relations should be equal, both updates will only use C_{b_1}. Thus, blocks b_1, b_2 are obtained from b in the following way: For each update $\varphi \to R \mathrel{+}= \bar{z}$ in b, the block b_1 contains the two updates

$$\begin{array}{l} \varphi \to R \mathrel{+}= \bar{z} \\ [\varphi]' \wedge \neg\mathit{Informed}(x) \to R' \mathrel{+}= \bar{z} \end{array}$$

and b_2 contains the update

$$[\varphi]' \wedge \mathit{Informed}(x) \to R' \mathrel{+}= \bar{z}$$

Removals $\varphi \to R \mathrel{-}= \bar{z}$ are handled analogously. Finally, the sequence β' of blocks specifies how the *Informed* relation is updated. Namely, for every statement $\varphi \to R \mathrel{+}= (x, \bar{y})$ or $\varphi \to R \mathrel{-}= (x, \bar{y})$, the sequence β' contains a corresponding block that handles the updates to the *Informed* relation:

$$\textbf{forall } x, \bar{y}.\ R\,x\,\bar{y} \not\leftrightarrow R'\,x\,\bar{y} \to \mathit{Informed} \mathrel{+}= (x)$$

Example 4. *Consider the block*

$$\textbf{forall } x, p, r \textbf{ may}.\ \mathit{Assign}(x, p) \to \mathit{Review} \mathrel{+}= (x, p, r)$$

Then the transformation results in the blocks

$$\begin{array}{l} \textbf{forall } x, p, r \textbf{ may}. \\ \quad \mathit{Assign}(x, p) \to \mathit{Review} \mathrel{+}= (x, p, r) \\ \quad \mathit{Assign}'(x, p) \wedge \neg\mathit{Informed}(x) \to \mathit{Review}' \mathrel{+}= (x, p, r) \\ \textbf{forall } x, p, r \textbf{ may}. \\ \quad \mathit{Assign}'(x, p) \wedge \mathit{Informed}(x) \to \mathit{Review}' \mathrel{+}= (x, p, r) \\ \textbf{forall } x, p, r. \\ \quad \mathit{Review}(x, p, r) \not\leftrightarrow \mathit{Review}'(x, p, r) \to \mathit{Informed} \mathrel{+}= (x) \end{array}$$

The fact that the two updates to Review′ are split into two blocks captures the semantics of causality. If x is informed, *the update will be done by the second block, and thus use C_{b_2} — a (possibly) different predicate than the one used for the update of Review. If not, the update will be done by the first block and use the same C_{b_1} predicate that is used for the update of Review.* □

The transformation for stubborn agents, denoted $\mathcal{T}_a^{(s)} w$, is similar. It consists only of the first two cases of the transformation for causal agents. For completeness, it is described in Appendix B.

The transformed workflow $\mathcal{T}_a^{(c)} w$ (or $\mathcal{T}_a^{(s)} w$) captures all pairs of traces of w that satisfy the causal (or stubborn) agent model together with declassification, relative to a.

Theorem 1. *Let w be a workflow. Then for each $m \in \{s, c\}$ and for every* FOLTL *formula ψ using predicates from $\mathcal{R}_{wf} \cup \mathcal{R}'_{wf}$ possibly mentioning a, the following statements are equivalent:*

- $w \models_2 \mathit{agent\_model}^{(m)} \to \forall a. \left(\mathsf{G}\, \mathit{eqOracles}(a)\right) \to \psi(a)$
- $\mathcal{T}_a^{(m)} w \models \psi(a)$

The proof is by establishing a simulation relation between states $s \otimes s'$ attained during a pair of traces of w satisfying $\mathit{agent\_model}^{(m)} \wedge \mathsf{G}\, \mathit{eqOracles}(a)$, and states $\bar{s}$ attained during a corresponding trace of $\mathcal{T}_a^{(m)} w$.

IV. Verification of Invariants

We have now transformed NDA into a universal invariant on the transformed workflow. In following sections we will now focus on the problem of proving that a universal invariant holds for a given workflow. In this section, we show that for any given universal invariant, we can prove or disprove inductiveness.

We let $\exists^*$FOL, $\forall^*$FOL, $\exists^*\forall^*$FOL, and $\forall^*\exists^*$FOL denote the fragments of first-order logic (FOL) consisting of those formulas whose prenex normal form equivalent has the quantifier prefix of the respective form.

We call a workflow w *guard-restricted* iff for each of its statements $\varphi \to R \mathrel{\pm}= \bar{u}$, the FOL formula φ is in the $\exists^*$FOL fragment. Guard-restrictedness will not affect the correctness of our methods. For workflows that are not guard-restricted, however, our methods will be incomplete. It is called *leveled* if each predicate can be assigned a *level* in $\mathbb{N}$ so that (1) the levels of all predicates updated in a block b agree, and (2) the level of R is less than the level of S whenever R occurs in the guard of an update to S. Intuitively, leveledness restricts how information flows from one predicate into another so that only predicates from lower levels may affect predicates on higher levels. Absence of leveledness, though, will not affect the correctness of our proposed methods. It rather will serve as one ingredient for a sufficient condition under which termination of our inference procedure can be guaranteed. In particular, the workflow from Fig. 2 is guard-restricted and leveled.

We remark that leveledness is a real restriction only in strongly connected components. For straight-line parts of the workflow, each update of a relation R that (in)-directly depends on R itself can be replaced by an update to a fresh relation R' that will be used to replace R in the following blocks.

Weakest Preconditions. Let w be a workflow and let ψ be any FOL formula with predicate symbols from $\mathcal{R}_{wf}$. For a block b, we define the *weakest precondition of ψ for b*, denoted $\mathsf{WP}[\![b]\!](\psi)$ by $\forall A_b.(\theta_b \psi)$ where A_b is a second-order variable. If b is a **may** block, then A_b is chosen as C_b. If b queries the oracle O, then $A_b = O$. Otherwise, we may choose A_b as any predicate symbol from $\mathcal{R}_c$, as then $\forall A_b.(\theta_b \psi)$ is equivalent to

$\theta_b\psi$. The transformation $\mathsf{WP}[\![b]\!]$ for blocks is readily extended to sequence β of blocks $b_h \ldots b_1$ by

$$\begin{array}{rcl}\mathsf{WP}[\![\beta]\!](\psi) & = & \mathsf{WP}[\![b_h]\!](\ldots\mathsf{WP}[\![b_1]\!](\psi)\ldots) \\ & = & \forall A_\beta.(\theta_\beta\,\psi)\end{array}$$

where A_β is the sequence $A_{b_h}, \ldots, A_{b_1}$ of second-order variables corresponding to the blocks in β. The next lemma formalizes the intuition behind the $\mathsf{WP}[\![\cdot]\!]$ function.

Lemma 1. *Let ρ be a valuation of the first-order variables in $\mathcal{X}$, let β be a sequence of blocks, and let s, s' be states so that $(s, s') \in \mathsf{next}_\beta$. Then $s', \rho \models \psi$ iff $s, \rho \models \mathsf{WP}[\![\beta]\!](\psi)$.*

It is particularly convenient when for all sequences β labeling edges in the workflow w, $\mathsf{WP}[\![\beta]\!]$ introduces no alternating first-order quantifiers. In this case, we call w *edge-uniform*. Edge-uniformity is, e.g., granted whenever the workflow is guard-restricted and has at most one block per edge (which is naturally provided by the standard representation of the workflow as a control flow graph). [1] Edge-uniformity is also preserved by the transformations introduced in Section III (Assuming that all declassification conditions are in $\exists^*$FOL). Therefore, it is a reasonable assumption in our context. A workflow is then called *uniform* if it is edge-uniform and also within each strongly connected component of w, every modified relation is either only modified by $+=$, or only by $-=$. Intuitively, uniformity implies that predicates will change only monotonically inside loops. Again, this restriction will not affect the correctness of our methods, but rather form the second ingredient of our sufficient condition for termination of the inference algorithm. The workflow from Fig. 2 is both edge-uniform and uniform.

Invariants. An *invariant* is an assignment Φ which maps each program point u of the workflow to a first-order formula $\Phi[u]$ over predicates in $\mathcal{R}_{wf}$ with free variables from $\mathcal{X}$. An invariant Φ is *universal* if each formula $\Phi[u]$ is in $\forall^*$FOL. It is *valid* if $\Phi[u]$ holds whenever program point u is reached on a trace starting in (u_0, s_0) for some initial state s_0, i.e., $s_0, \rho \models \varphi_0$. The invariant Φ of w is *inductive* iff for every edge (u, β, v) of the workflow, $\Phi[u] \to \mathsf{WP}[\![\beta]\!](\Phi[v])$ holds. Then an inductive invariant Ψ is valid iff $\varphi_0 \to \Psi[u_0]$ holds.

Theorem 2. *Let w be an edge-uniform workflow and Φ a universal invariant. It is decidable whether or not Φ is inductive and $\varphi_0 \to \Phi[u_0]$ holds.*

We remark that the restriction to *edge-uniform* workflows can be lifted by means of the abstraction techniques provided in the next section — leading to an incomplete verification method for universal invariants on unrestricted workflows.

Proof. Equivalently, we check whether $\varphi_0 \wedge \neg\Phi[u_0]$ as well as $\Phi[u] \wedge \neg\mathsf{WP}[\![\beta]\!](\Phi[v])$ are unsatisfiable, for each edge (u, β, v) of the workflow. We focus on the latter formula, as the other case is similar. Since w is edge-uniform and $\Phi[v]$ is a universal formula, $\mathsf{WP}[\![\beta]\!](\Phi[v])$ is of the form $\forall A_\beta.\forall\bar{y}.\psi$ where ψ has only existential first-order quantifiers, i.e., is contained in $\exists^*$FOL. Therefore, $\neg\mathsf{WP}[\![\beta]\!](\Phi[v])$ is given by

$$\exists A_\beta.\exists\bar{y}.\neg\psi$$

where $\neg\psi$ is in $\forall^*$FOL. Since the only second-order quantifiers in $\Phi[u] \wedge \neg\mathsf{WP}[\![\beta]\!](\Phi[v])$ are the existential quantifiers for the variables in A_β, satisfibility of this formula is equivalent with satisfiability of a first order-formula consisting of two conjuncts, one in $\forall^*$FOL (namely, $\Phi[u]$) and the other in $\exists^*\forall^*$FOL. So its unsatisfiability can be effectively checked [5]. □

V. Inferring Inductive Invariants

Unfortunately, not every invariant is naturally inductive. Thus, in this section we show how to compute a strengthening of a given universal invariant that is inductive.

Assume that we want to verify an assignment Ψ_0 of program points to assertions for an arbitrary workflow w, i.e., verify that $\Psi_0[u]$ holds whenever the program point u of the workflow w is reached. Such a certificate can be obtained from an inductive invariant Ψ of w so that

(a) $\Psi[u] \to \Psi_0[u]$ for all program points u; and
(b) $\varphi_0 \to \Psi[u_0]$ for the start point u_0 of w.

In case that $\Psi_0[u] = \psi$ for all u, we then have verified that $\mathsf{G}\,\psi$ holds for the given workflow w. The required inductive invariant Ψ (if it exists) may be complicated and not easy to guess. As in the last section, we restrict ourselves to *universal* inductive invariants using predicates of the workflow only. This means that we are interested in the refined question: Can the assignment Ψ_0 be certified by a universal inductive invariant?

In light of requirement (b), it thus suffices to determine the *weakest* universal inductive invariant $\bar{\Psi}$ satisfying requirement (a). Given that this invariant exists and is computable, then the assignment Ψ_0 can be certified by means of a universal inductive invariant iff $\varphi_0 \to \bar{\Psi}[u_0]$.

In order to determine $\bar{\Psi}$, we put up the constraint system $\mathcal{C}$:

$$X[u] \to \Psi_0[u] \qquad (1)$$

$$X[u] \to \mathsf{WP}[\![\beta]\!](X[v]) \quad \text{if } (u, \beta, v) \text{ is an edge of } w \qquad (2)$$

where the unknown $X[u]$ represents the potential formula assigned to program point u. By definition, any assignment X satisfying all constraints (2) is inductive where property (a) for X is expressed by the additional constraint (1). The set of universal formulas (modulo semantic equivalence) forms a *lattice*, when implication is seen as the ordering relation $\sqsubseteq$. W.r.t. this ordering, the greatest and least elements $\top$ and $\bot$ are represented by the formulas *true* and *false*, respectively. Likewise, the greatest lower bound of a finite set of formulas is given by their conjunction. The lattice is, however, not a *complete* lattice, i.e., not all sets of formulas necessarily have a greatest lower bound. In particular, it may have infinite

[1] A scenario where there is more than one block per edge corresponds to the situation where we are interested in properties only at specific program points.

decreasing chains — at least if there are two or more binary predicates E and T. To see this, consider the formulas

$$\begin{array}{rl} c_k = & (E(x_0,x_1)\wedge\ldots\wedge E(x_{k-1},x_k)) \to T(x_0,x_k)\\ \varphi_k = & \forall x_0,\ldots,x_k.\ c_0\wedge\ldots\wedge c_k \end{array}$$

for $k\geq 1$. Then all formulas φ_k are pairwise inequivalent, while at the same time $\varphi_{k+1}\to\varphi_k$ for all $k\geq 1$ holds. In general, it is thus not guaranteed that a greatest solution (corresponding to the weakest inductive invariant) exists. In order to come up with practical means for solving system $\mathcal{C}$ at least in some cases, we consider two useful constructions, namely an abstraction technique of existential (first-order) quantifiers, and an algorithm for eliminating second-order universal quantifiers. These two techniques will allow us to further simplify the right-hand sides in the constraints of $\mathcal{C}$.

A. Approximating First-Order Existential Quantification

Consider a sequence of blocks β occurring in an edge label of the workflow w. Then the substitution θ_β may introduce fresh existential as well as fresh universal quantifiers. Since we are interested in universal inductive invariants only, our goal is to systematically remove the occurring existential quantifiers. We find:

Theorem 3. *For every formula ψ with free variables from $\mathcal{X}$, a formula $\psi^\sharp$ can be constructed using universal quantification only so that the following two properties are satisfied:*

(1) $\psi^\sharp\to\psi$;

(2) If ψ is in $\forall^*\exists^*$FOL, then $\varphi\to\psi^\sharp$ holds for every universal formula φ with free variables from $\mathcal{X}$ such that $\varphi\to\psi$.

In light of the second statement of Theorem 3, the formula $\psi^\sharp$ can be seen as the uniquely determined weakest strengthening of formulas ψ in $\forall^*\exists^*$FOL to a universal formula. The formula $\psi^\sharp$ is called the *universal abstraction* of ψ.

Proof. W.l.o.g., let us assume that ψ is in negation normal form (i.e., using conjunction and disjunction as only boolean connectives, and negation only applied to atomic propositions), and that nested universally bound variables are distinct.

We proceed by induction on the structure of ψ. For that, we introduce a transformation $[\,.\,]^\sharp_{X'}$ on subformulas of ψ, for a set X' of variables intended to be those free variables in the current subformula which are universally quantified in ψ. Then $\psi^\sharp$ is defined as $[\psi]^\sharp_{\mathcal{X}}$. The transformation $[\,.\,]^\sharp_{X'}$ is defined as follows:

$$\begin{array}{lll} [\forall x.\psi']^\sharp_{X'} & = & \forall x.[\psi']^\sharp_{X'\cup\{x\}}\\ [\exists y.\psi']^\sharp_{X'} & = & \bigvee_{x\in X'}[\psi']^\sharp_{X'}[x/y]\\ [\psi_1\vee\psi_2]^\sharp_{X'} & = & [\psi_1]^\sharp_{X'}\vee[\psi_2]^\sharp_{X'}\\ [\psi_1\wedge\psi_2]^\sharp_{X'} & = & [\psi_1]^\sharp_{X'}\wedge[\psi_2]^\sharp_{X'}\\ [\psi']^\sharp_{X'} & = & \psi' \qquad \text{otherwise} \end{array}$$

We claim that $\psi^\sharp$ implies ψ. For that, we prove for each finite subset X' of variables X' and each subformula ψ', that $[\psi']^\sharp_{X'}\to\psi$ holds. The proof is by induction on the structure of ψ'. The only interesting case is when ψ' is of the form $\exists y.\,\psi''$. By induction hypothesis, $[\psi'']^\sharp_{X'}\to\psi''$ holds. Thus, also $[\psi'']^\sharp_{X'}[x/y]\ \to\exists y.\,\psi''$ holds for each $x\in X'$. From that the claim follows for ψ'.

For statement (2), consider any formula φ with $\varphi\ \to\ \psi$. Then $\varphi\wedge\neg\psi$ is unsatisfiable. Let ψ equal $\forall x_1,\ldots,x_r.\exists y_1,\ldots,y_s.\psi'$ with ψ' quantifier-free. Then

$$\exists x_1\ldots x_r.\,(\varphi\wedge\forall y_1\ldots y_s.\neg\psi')$$

must be unsatisfiable. Since that formula is equisatisfiable with

$$\exists x_1\ldots x_r.\,(\varphi\wedge\textstyle\bigwedge_{i_1=1}^{r}\cdots\bigwedge_{i_s=1}^{r}\neg\psi'[x_{i_1}/y_1,\ldots,x_{i_s}/y_s])$$

this implies that also $\varphi\wedge\neg(\psi^\sharp)$ is unsatisfiable. Consequently, $\varphi\to\psi^\sharp$ holds. □

Applying Theorem 3 to our setting, we find:

Corollary 1. *Assume that β is a sequence of blocks so that $\mathsf{WP}[\![\beta]\!](\psi')$ is of the form $\forall A_h,\ldots,A_1.\psi''$ for some formula ψ'' in $\forall^*\exists^*$FOL. Then $\psi\to\mathsf{WP}[\![\beta]\!](\psi')$ iff $\psi\to\forall A_h,\ldots,A_1.(\psi'')^\sharp$ holds.*

Proof. It suffices to prove that $\psi\wedge\psi''$ is unsatisfiable iff $\psi\wedge(\psi'')^\sharp$ is unsatisfiable. That, however, follows from Theorem 3. □

We remark that the abstraction function $(\ldots)^\sharp$ commutes with substitutions.

Lemma 2. *Let ψ,φ be formulas, R a predicate of arity r, and $\bar{x}$ the sequence of variables $x_1\ldots x_r$. Let θ_1,θ_2 denote the substitutions $R\bar{x}\mapsto R\bar{x}\vee\varphi$ and $R\bar{x}\mapsto R\bar{x}\wedge\neg\varphi$. Then the following holds:*

1) $(\theta_1\psi)^\sharp=(\theta_1(\psi^\sharp))^\sharp$;

2) $(\theta_2\psi)^\sharp=(\theta_2(\psi^\sharp))^\sharp$. □

When universal invariants are of concern, Lemma 2 implies that it does not make a difference whether we abstract first and apply substitutions later, or postpone the abstraction to the very end.

We remark that the given technique for strengthening formulas with existential quantifiers by means of universal formulas only, can be applied to prove a given universal invariant inductive — irrespective of whether the workflow is edge-uniform or not.

B. Eliminating Second-Order Universal Quantification

Now consider a sequence of blocks β occurring in an edge label of the workflow w, which is either a **may** block or queries an oracle. Then the weakest precondition $\mathsf{WP}[\![\beta]\!]$ introduces universal quantification over some predicates A_β, not mentioned by the workflow and thus also not mentioned by any of the formulas assigned to program points u or v. In the following, we provide a method for eliminating such second-order quantifications.

Fact 1. *The clause*

$$\forall A.\ A\bar{z}_1\vee\ldots\vee A\bar{z}_r\vee\neg A\bar{z}'_1\vee\ldots\vee\neg A\bar{z}'_s \tag{3}$$

for sequences $\bar{z}_i, \bar{z}'_j$ of variables is equivalent to

$$\bigvee_{i=1}^{r} \bigvee_{j=1}^{s} \bar{z}_i \doteq \bar{z}'_j \tag{4}$$

where the equality between sequences of variables equals the conjunction of the equalities between corresponding variables.

Proof. Let us fix some values for the occurring first-order variables. First assume that the formula (4) holds (w.r.t. that variable assignment). Then there are some i, j so that the conjunction of equalities $\bar{z}_i \doteq \bar{z}'_j$ holds. Then $A\bar{z}_i \vee \neg A\bar{z}'_j$ is equivalent to *true* for every predicate A. Therefore, formula (3) holds as well. For the reverse implication, assume that (4) does not hold for the given variable assignment. Then for all i, j, the sequences z_i, z'_j are different. Then some predicate A exists so that $A\bar{z}_i$ is *false* for all i, and $A\bar{z}'_j$ is *true* for all j. For that particular predicate A and the given variable assignment, the clause $A\bar{z}_1 \vee \ldots \vee A\bar{z}_r \vee \neg A\bar{z}'_1 \vee \ldots \vee \neg A'\bar{z}'_s$ is *false*. Therefore, formula (3) evaluates to *false* as well, thus proving the reverse implication. □

Universal quantification generally satisfies the following laws:

$$\forall A.\ \varphi_1 \wedge \varphi_2 = \qquad (\forall A.\varphi_1) \wedge (\forall A.\varphi_2) \tag{5}$$

$$\forall A.\ \varphi_1 \vee \varphi_2 = \ \varphi_1 \vee (\forall A.\ \varphi_2) \text{ if } A \text{ does not occur in } \varphi_1 \tag{6}$$

Therefore, Fact 1 gives rise to an effective second-order quantifier elimination.

Theorem 4. *Assume that ψ is a quantifier-free formula. Then a quantifier-free formula ψ' can be constructed so that $\psi' \leftrightarrow \forall A.\, \psi$ holds.*

Proof. Assume, w.l.o.g., that ψ is in conjunctive normal form. Since universal quantification distributes over conjunctions, we may apply quantifier elimination to each clause c of ψ separately. Any given clause c can be written as $c_1 \vee c_2$ where c_1 does not contain occurrences of A and c_2 collects all literals with predicate A. Then $\forall A.\ c$ is equivalent to $c_1 \vee c'_2$ where c'_2 is determined from c_2 according to Fact 1. This completes the construction. □

The proposed procedure for second-order quantifier elimination is not new (Isabelle, e.g., easily verifies Fact 1). Implicitly, our procedure can be considered as a particular instance of the SCAN algorithm [12], [14].

Let ψ be quantifier-free and in conjunctive normal form. A predicate A *covers* another predicate A' in ψ if the following two properties are met by all clauses c of ψ:

1) Whenever c has a literal $A'\bar{z}$, then c also has a literal $A\bar{z}$;
2) Whenever c is of the form $c' \vee \neg A'\bar{z}$, then $c' \vee \neg A\bar{z}$ is also a clause of ψ.

Lemma 3. *Let A, A' be two predicates in ψ so that A covers A'. Let ψ' be obtained from ψ by removing all literals $A'\bar{z}$ and all clauses containing $\neg A'\bar{z}$. Then $\forall\, A, A'.\ \psi$ is equivalent to $\forall\, A.\ \psi'$.*

Proof. The idea is that all conjunctions of equalities $\bar{z} \doteq \bar{z}'$ introduced into clauses by removal of universal quantification over A' are already introduced by universal quantification over A. □

C. The Fixpoint Iteration

In light of the constructions from the last two subsections, we introduce the abstracted constraint system $\mathcal{C}^\sharp$ consisting of the constraints:

$$\begin{array}{lcll} X[u] & \rightarrow & \Psi_0[u] & \\ X[u] & \rightarrow & \forall A_\beta(\theta_\beta X[v])^\sharp & \quad \text{if } (u, \beta, v) \text{ is an edge of } w \end{array}$$

According to our assumptions, $\Psi_0[u]$ is a universal formula using the predicates from the workflow w only. By applying the abstraction of existentials, followed by second-order quantifier elimination, each evaluation of a right-hand side of $\mathcal{C}^\sharp$ on a given assignment X returns a universal first-order formula. For $h \geq 0$, let $X^{(h)}$ denote the assignment of program points to formulas which is attained after h rounds of fixpoint iteration, i.e., $X^{(0)}[u] = \Psi_0[u]$, and for $h > 0$,

$$X^{(h)}[u] = \Psi_0[u] \wedge \bigwedge \{\forall A_\beta.\ (\theta_\beta(X^{(h-1)}[v]))^\sharp \mid (u, \beta, v) \text{ edge}\}$$

For a block b, let $\theta_b^{(j)}$ denote the substitution corresponding to b where the predicate variable A_b is substituted with the predicate A_j (of appropriate arity). Moreover, let $\Pi^{(h)}[u, v]$ denote the set of sequences of blocks occurring on paths of length at most h starting from u and reaching v. Then we have

Lemma 4. *For every $h \geq 0$ and every program point u,*

$$\begin{array}{rcl} X^{(h)}[u] & = & \bigwedge \{\forall A_m \ldots A_1 . (\theta_{b_m}^{(m)} \ldots (\theta_{b_1}^{(1)}\, \Psi_0[v]) \ldots)^\sharp \mid \\ & & \qquad b_m \ldots b_1 \in \Pi^{(h)}[u, v]\} \end{array} \tag{7}$$

Proof. The proof is by induction on h. For $h = 0$, $\Pi^{(0)}[u, u]$ consists of ϵ only, and $\Pi^{(0)}[u, v] = \emptyset$ for $u \neq v$. Since $X^{(0)}[u] = \Psi_0[u]$, the claim follows. Now assume that the assertion is true for $h - 1$. We have the sequence of equalities given in Figure 4, where equality $(*)$ follows by distributivity of substitutions and $(\ldots)^\sharp$ with $\wedge$ and compatibility of $(\ldots)^\sharp$ with substitutions. □

Lemma 4 assures that after h rounds of fixpoint iteration, a formula at u is attained which is the weakest universal precondition of Ψ_0 w.r.t. paths of length at most h. Furthermore, due to Theorem 4, all second-order quantifiers therein can be eliminated. Thus, the only reason why fixpoint iteration for constraint system $\mathcal{C}^\sharp$ may not terminate, is that an ever growing number of first-order universally quantified variables is introduced. We obtain:

Theorem 5. *Assume that w is a workflow, and Ψ_0 is an initial assignment of program points to universal formulas using predicates from w only. Assume further that during the fixpoint iteration for the constraint system $\mathcal{C}^\sharp$ only finitely many universally quantified first-order variables are introduced. Then the iteration terminates with an assignment Ψ such that the following holds:*

(1) Ψ is a universal inductive invariant of w with $\Psi[u] \rightarrow \Psi_0[u]$ for all program points u of w.

$$
\begin{aligned}
X^{(h)}[u] &= \Psi_0[u] \wedge \bigwedge\{\forall A_{b_1} \ldots A_{b_\mu}.(\theta_{b_1}(\ldots(\theta_{b_\mu}\, X^{(h-1)}[u'])\ldots))^\sharp \mid (u, b_1 \ldots b_\mu, u') \text{ edge}\} \\
&= \Psi_0[u] \wedge \bigwedge\{\forall A_1 \ldots A_\mu.(\theta^{(1)}_{b_1}(\ldots(\theta^{(\mu)}_{b_\mu}\, X^{(h-1)}[u'])\ldots))^\sharp \mid (u, b_1 \ldots b_\mu, u') \text{ edge}\} \\
&= \Psi_0[u] \wedge \bigwedge\{\forall A_1 \ldots A_\mu.(\theta^{(1)}_{b_1}(\ldots(\theta^{(\mu)}_{b_\mu} \bigwedge\{\forall A_1 \ldots A_m.(\theta^{(1)}_{b_{\mu+1}}(\ldots(\theta^{(m)}_{b_{\mu+m}}\, \Psi_0[v])\ldots))^\sharp \mid \\
&\qquad b_{\mu+1} \ldots b_{\mu+m} \in \Pi^{(h-1)}[u', v]\})\ldots))^\sharp \mid (u, b_1 \ldots b_\mu, u') \text{ edge}\} \qquad \text{by IH} \\
&= \Psi_0[u] \wedge \bigwedge\{\forall A_1 \ldots A_\mu.(\theta^{(1)}_{b_1}(\ldots(\theta^{(\mu)}_{b_\mu} \bigwedge\{\forall A_{\mu+1} \ldots A_{\mu+m}.(\theta^{(\mu+1)}_{b_{\mu+1}}(\ldots(\theta^{(\mu+m)}_{b_{\mu+m}}\, \Psi_0[v])\ldots))^\sharp \mid \\
&\qquad b_{\mu+1} \ldots b_{\mu+m} \in \Pi^{(h-1)}[u', v]\})\ldots))^\sharp \mid (u, b_1 \ldots b_\mu, u') \text{ edge}\} \\
&= \Psi_0[u] \wedge \bigwedge\{\forall A_1 \ldots A_{\mu+m}.(\theta^{(1)}_{b_1}(\ldots(\theta^{(\mu+m)}_{b_{\mu+m}}\, \Psi_0[v])\ldots))^\sharp \mid \\
&\qquad b_{\mu+1} \ldots b_{\mu+m} \in \Pi^{(h-1)}[u', v], (u, b_1 \ldots b_\mu, u') \text{ edge}\} \qquad (*) \\
&= \Psi_0[u] \wedge \bigwedge\{\forall A_1 \ldots A_{\mu+m}.(\theta^{(1)}_{b_1}(\ldots(\theta^{(\mu+m)}_{b_{\mu+m}}\, \Psi_0[v])\ldots))^\sharp \mid b_1 \ldots b_{\mu+m} \in \Pi^{(h)}[u, v]\}
\end{aligned}
$$

Figure 4. Equalities used in the proof of Lemma 4.

(2) If w is guard-restricted, then Ψ is the weakest *assignment with property (1).*

Still, it is desirable to have widely applicable structural conditions which are sufficient for guaranteeing termination of fixpoint iteration.

VI. Termination

In this section we show that fixpoint iteration for the constraint system $\mathcal{C}^\sharp$ terminates for workflows which are both *uniform* and *leveled*. Intuitively, this sufficient condition means that inside strongly connected components of the control flow graph for w, each predicate either only increases or only decreases, and no guard of such a modification of a predicate R, directly or indirectly depends on R itself.

Termination of fixpoint iteration even for uniform and leveled workflows is by no means trivial, as each application of the weakest precondition operator may introduce further universally quantified variables. Here is our main technical theorem.

Theorem 6. *Assume that the workflow w is uniform and leveled, and Ψ_0 is an initial assignment of program points to universal formulas using predicates of w only. Then there is a weakest universal inductive invariant Ψ with $\Psi[u] \to \Psi_0[u]$ for all program points of w. Moreover, Ψ can be effectively computed.*

Proof. Every inductive invariant Ψ' so that $\Psi'[u] \to \Psi_0[u]$ for all program points u, is a solution of the constraint system $\mathcal{C}^\sharp$.

By Lemma 4, the formula $X^{(h)}$ is equivalent to a conjunction of formulas $\forall A_{h'} \ldots A_1.((\theta_{h'} \circ \ldots \circ \theta_1)\Psi_0[v])^\sharp$. By Theorem 7, each such formula is equivalent to a similar formula, where the number of applied substitutions is bounded by some fixed number M. Up to logical equivalence, there are only finitely many of such formulas. Accordingly, for each program point u and each $h \geq 0$, there are only finitely many possible values for $X^{(h)}[u]$. Therefore, for some $h \geq 0$, $X^{(h+1)}[u] \leftrightarrow X^{(h)}[u]$ for all program points u. In that case, $X^{(h)}$ represents the greatest solution of $\mathcal{C}^\sharp$ and therefore is the desired weakest inductive invariant. □

Theorem 6 does not require the workflow w to be guard-restricted. In case that w makes use of arbitrary first-order quantification in guards, the algorithm still is guaranteed to terminate. However, it will terminate with some universal inductive invariant — not necessarily the weakest.

Assume that the set of predicates $\mathcal{R}_{wf}$ comes together with an assignment of levels. Assume further that we are given a finite set $\mathcal{B}$ of blocks b, each of which gives rise to a substitution θ_b of a subset of predicates which are all of the same level i. In that case, we say that i is the level of b. Let Θ denote the set of all these substitutions. For $i \geq 1$, let A_i denote a sequence of fresh distinct predicate names. For a sequence $\beta = b_h \ldots b_1$ of blocks, let $\theta_\beta = \theta_h \circ \ldots \circ \theta_1$ where θ_i is obtained from the substitution θ_{b_i} for block b_i by replacing the predicate A_{b_i} (if present) with A_i.

Theorem 7. *Let m be the maximal number of distinct substitutions in Θ. Furthermore, let r denote the number of levels of predicates in $\mathcal{R}_{wf}$. Assume that ψ is a first-order formula with predicates from $\mathcal{R}_{wf}$. Assume that $\theta_\beta = \theta_h \circ \ldots \circ \theta_1$ is a sequence of substitutions corresponding to a sequence of blocks β of length h. Then there is a sub-sequence $\theta' = \theta_{j_1} \circ \ldots \circ \theta_{j_l}$ of θ_β with $l < (1 + \frac{m}{r})^r$ such that the following equivalence holds.*

$$\forall A_1 \ldots A_s.\, (\theta_\beta \psi)^\sharp \quad \equiv \quad \forall A_{j_1} \ldots A_{j_l}.\, (\theta' \psi)^\sharp \tag{8}$$

The proof is based on the observation that the substitution corresponding to the same block b may produce only finitely often new contributions to the composition — at least, if the corresponding second-order quantifiers are later eliminated. The full proof can be found in Appendix C.

VII. Application to Noninterference

As a warm-up, let us consider stubborn agents only. Assume that the guard-restricted workflow w is uniform and leveled. Then the same is true for the transformed workflow $\mathcal{T}^{(s)}_a w$ which takes care of stubbornness of agents and declassification relative to a. Accordingly, we obtain as applications of Theorem 6:

Theorem 8. *Consider a workflow w which is uniform and leveled, and assume that all agents participating in w are stubborn. Assume furthermore that all declassification predicates $declass_O(a, \bar{x})$ are quantifier-free. Let a be some agent variable and Ψ_0 an assignment of the nodes of w to universal* *formulas using predicates from $\mathcal{R}_{wf} \cup \mathcal{R}'_{wf}$ and free variables from $\{a\} \cup \mathcal{X}$. Then*

(1) $\mathcal{T}_a^{(s)} w$ is again uniform and leveled.

(2) A universal inductive invariant Ψ of $\mathcal{T}_a^{(s)} w$ can be effectively constructed so that $\Psi[u] \to \Psi_0[u]$ for all program points u.

(3) If w is guard-restricted then the invariant Ψ from (2) can be chosen as the weakest universal invariant with that property.

Corollary 2. *For a given workflow w which is uniform and leveled, it is decidable whether there exists a universal invariant that implies NDA of w for $agent\_model^{(s)}$.*

We would like to apply the same strategy to certify noninterference also for causal agents. Again assume that the workflow w is uniform and leveled. The workflow $\mathcal{T}_a^{(c)} w$ then, however, is no longer leveled. This is due to the auxiliary informedness predicate $Informed$ introduced by $\mathcal{T}_a^{(c)}$. That predicate is queried at the update of every predicate R', $R \in \mathcal{R}_{wf}$, and likewise its update is guarded by formulas which also depend on R'. Still, we then can apply the given fixpoint iteration — which, however, is no longer guaranteed to terminate. Thus, we obtain an incomplete method to certify non-interference for agent model $agent\_model^{(c)}$. The potential non-termination, however, does not come as a surprise as noninterference in general is undecidable for an unbounded number of causal agents [10]. Interestingly, the situation is different when only a *fixed* bounded number of agents behaves causally, while all others behave stubbornly.

Assume that at most $k \geq 0$ behave causally, while all other agents are stubborn. Our goal is again to modify the workflow in order to take care of the agent model and declassification. In case of at most k causal agents, the informedness predicate may receive only finitely many values. That finite value therefore can be encoded into the program points of the transformed workflow. Updates to informedness then show up as *guards* as last actions at a control flow edge. These thus now have the form $(u, \beta g, v)$ for some first-order formula g — with the intended semantics that the edge can only be taken if the state s attained after executing the blocks in β, satisfies g, i.e., $s, \rho \models g$. Accordingly, the weakest precondition operator for g is defined by $\mathsf{WP}[\![g]\!](\psi) = \neg g \vee \psi$.

Let $y_1, \ldots, y_k$ denote a sequence of k distinct fresh variables. Consider a block b of w, of an edge (u, β, v) of w. For simplicity, we assume that b is the only block at this edge, i.e., $\beta = b$. Let $Y, Y' \subseteq \{y_1, \ldots, y_k\}$, $Y \subseteq Y'$, denote the subsets of agents which are informed before and after executing the block b, respectively.

First, assume that b is not a **may** block. Then $\mathcal{T}_{Y,Y'}^{(c)} b$ is defined analogously to $\mathcal{T}^{(c)} b$ — with the major difference that now a guard $g_{Y,Y'}$ is introduced to take care of the required update of informedness. Thus, $\mathcal{T}_{Y,Y'}^{(c)} b = \beta g_{Y,Y'}$ where the sequence of (one or two) blocks β is defined analogously to the corresponding blocks in $\mathcal{T}^{(c)}$, and the guard $g_{Y,Y'}$ is the conjunction of formulas

$$y_j \in Y \vee \exists\, \bar{z}. R y_j \bar{z} \not\leftrightarrow R' y_j \bar{z}$$

for all $y_j \in Y'$ and predicates R updated in the block b. Here, the expression $y_j \in Y$ is a shortcut for $\bigvee_{y_i \in Y} y_j \doteq y_i$.

Now assume that b is of the form **forall** $x, \bar{x}'$. **may** *stmts*. Then $\mathcal{T}_{Y,Y'}^{(c)} b = b_1 b_2 g_{Y,Y'}$ where the blocks b_i, with $i \in \{1, 2\}$, are of the form **forall** $x, \bar{x}'$. **may** *stmts*$_i$ where for every update $\varphi \to R \mathrel{+}= \bar{z}$ of *stmts*, *stmts*$_1$ has the statements

$$\begin{array}{l} \varphi \to R \mathrel{+}= \bar{z} \\ [\varphi]' \wedge (x \notin Y) \to R' \mathrel{+}= \bar{z} \end{array}$$

and *stmts*$_2$ has the statement

$$[\varphi]' \wedge (x \in Y) \quad \to \quad R' \mathrel{+}= \bar{z}$$

An update $\varphi \to T \mathrel{-}= \bar{z}$ is treated analogously. Here, the guard $g_{Y,Y'}$ is defined identically as for non-**may** blocks.

The new workflow $\mathcal{T}^{(c,k)} w$ then consists of all control flow edges

$$(\langle u, Y \rangle, \mathcal{T}_{Y,Y'}^{(c)} \beta, \langle u, Y' \rangle)$$

for edges (u, β, v) of w and $Y, Y' \subseteq \{y_1, \ldots, y_k\}$ with $Y \subseteq Y'$. The size of the resulting workflow has increased by a factor of 2^k. All occurrences of the predicate $Informed$, on the other hand, have disappeared — implying that $\mathcal{T}^{(c,k)} w$ is leveled whenever w is leveled. Moreover, all guards g occurring in $\mathcal{T}^{(c,k)} w$ are contained in $\exists^*$FOL, implying that the transformation preserves edge-uniformity. The correctness of the transformation can be proven along the same lines as Theorem 1. Theorem 6 can still be applied, since the only guards introduced inside strongly connected components are of the form $g_{Y,Y}$ — and thus always equivalent to $true$. Therefore, we finally obtain:

Theorem 9. *Consider a workflow w which is uniform and leveled, and assume that $k \geq 0$ of the agents participating in w are causal, while all other agents are stubborn. Assume furthermore that all declassification predicates $declass_O(a, \bar{x})$ are quantifier-free. Then for some agent variable a and an assignment Ψ_0 of universal formulas using predicates from $\mathcal{R}_{wf} \cup \mathcal{R}'_{wf}$ and free variables from $\{a\} \cup \mathcal{X}$ to the nodes of w, the following holds.*

(1) $\mathcal{T}_a^{(c,k)} w$ is again uniform and leveled.

(2) A universal inductive invariant Ψ of $\mathcal{T}_a^{(c,k)} w$ can be effectively constructed so that $\Psi[u] \to \Psi_0[u]$ for all program points u.

(3) If w is guard-restricted, the invariant Ψ from (2) can be chosen as the weakest universal invariant with that property.

Corollary 3. *For every $k \geq 0$ and a given workflow w which is uniform and leveled, it is decidable whether there exists a universal invariant that implies NDA of w for $agent\_model^{(c,k)}$.*

VIII. Experimental Evaluation

We have implemented our approach into the tool NIWO-invariants. The source code together with all examples can be found on the authors' website . As input the tool takes the specification of a workflow together with declassification conditions and the agent model where we consider all stubborn or all causal agents only. It then encodes agent model and declassification conditions into the workflow and solves the corresponding constraint system $\mathcal{C}^\sharp$. The resulting verification conditions in $\exists^*\forall^*$FOL are then strengthened further by replacing equalities with *false*. This allows to encode the syntactically occurring literals as propositional variables and thus reduce the verification conditions to boolean satisfiability problems. These are then checked by Z3 [8].

Experiments. We used several variations of a conference management system to showcase the properties of our approach. *Conference_linear* is the motivating example used in [11]. It is a simpler version of the workflow from Example 2, which does not use loops, and already exhibits an attack for causal agents. Thus, no invariant exists that proves NDA when considering causal agents. *Conference_linear_fixed* is the fixed version presented by the authors. It is a naturally *omitting* workflow, which could not be dealt with automatically by previous work. It can now be proven safe by our tool. *Conference_omitting* is the omitting workflow from Example 2. Our implementation proves it safe for stubborn agents. It cannot be proven safe for causal agents, as there is a possible attack. *Conference_omitting_fixed* is a modification which excludes the given attack. This example is again omitting, and so could not be dealt with automatically so far. Our tool is able to prove it safe. *Conference_nonomitting* is the motivating example used in [10]. It is an non-omitting variant of Example 2, and accordingly less realistic. It also exhibits an attack for causal agents, but can be proven safe for stubborn ones. All these workflows except the fixed versions, are uniform and leveled.

Results. Our tool is able to prove safe all examples that do not exhibit attacks. Interestingly, even though termination is not guaranteed for causal agents or for non-leveled workflows, our tool still terminated on all examples we considered. The results of the experiments are shown in Fig. 5. The first columns give workflow type (omitting/non-omitting), the size of the workflow (the number of blocks the workflow consists of, not counting choice and loop constructs), and the considered agent model. The result is marked as *valid* iff our tool could find a strengthened universal inductive invariant that implies NDA and it is marked as *invalid* otherwise. The number of strengthenings (column #Str.) is the maximal number of updates of assertions at program points. The size of the largest/average assertion (columns Max./Avg. inv.) is the number of nodes in the formulas' abstract syntax trees. The last column reports the time (in milliseconds) for checking validity (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. As expected of a modern satisfiability solver, Z3 was able to check the satisfiability of our formulas easily even though the size of the resulting boolean formula is exponential in the maximum universal quantifiers per block and the number of strengthenings needed. For stubborn agents, all examples terminated after at most 2.5 seconds. For causal agents, the number of strengthenings increased and invariants became significantly larger. Still, all examples terminated within at most 22 seconds.

Comparison to other tools. The only tool for automatic workflow verification we are aware of is NIWO, described in [10]. It implements a procedure to verify NDA by compiling it to an equisatisfiable LTL formula to be checked by an off-the-shelf LTL satisfiability solver. As it is not able to deal with omitting workflows, it can only be applied to a more restricted class of workflows. It is also not able to deal with the agent model where all agents are causal, but only applicable to a fixed number of causal agents. In contrast, our tool is able to find a universal invariant that implies NDA. In addition, our tool is faster than NIWO on all examples they can both be applied to. While NIWO takes several seconds to several minutes to solve even comparatively simple examples, our tool handled all examples in a fraction of the time.

IX. Related Work

The work closest to ours is [10], which we have already discussed in the introduction and throughout the paper. The workflow model for web-based systems that we use has been introduced in [11]. That workflow language did not have a loop construct and it thus enabled a bounded model checking approach for verification. Generally, there is a growing interest in verifying infinite-state or parametric systems via a formalization in first-order logic. One such attempt is the programming language Ivy [20], which has been used to model and check a variety of parameterized systems, for instance the Paxos protocol [19]. Ivy is similar to the workflow language considered here in that its only data structure are finite relations. As in our work, the language restricts its statements in such a way that checking whether universal invariants are inductive reduces to checking the satisfiability of an $\exists^*\forall^*$FOL formula. However, in contrast to our work, Ivy restricts the control flow to a single loop. Furthermore, there is no attempt to infer invariants. Another recent verification approach is present in the VeriCon system [2], which has been proposed for describing and verifying the semantics of controllers in software-defined networks. The semantics of the underlying language is also specified, as in our work, in terms of relations. However, as detailed in [10], the semantics in [2] and that of workflows differ and cannot easily simulate one another. Network invariants are checked with Z3 and iteratively strengthened if they are not inductive — without providing termination guarantees.

First-order transition systems have also been used in other domains, such as AI, where an application is to model reachability problems that arise in robot planning. For instance,

Name	Type	Size	Model	Result	#Str.	Max. inv.	Avg. inv.	Time
Conference_linear	non-omitting	4	stubborn	valid	3	179	75	338 ms
Conference_linear	non-omitting	4	causal	invalid	3	1670	562	1537 ms
Conference_linear_fixed	omitting	5	stubborn	valid	4	247	84	322 ms
Conference_linear_fixed	omitting	5	causal	valid	4	4285	1107	2352 ms
Conference_omitting	omitting	6	stubborn	valid	5	220	81	347 ms
Conference_omitting	omitting	6	causal	invalid	7	2615	1066	2924 ms
Conference_omitting_fixed	omitting	7	stubborn	valid	6	709	202	1059 ms
Conference_omitting_fixed	omitting	7	causal	valid	8	9226	2132	5450 ms
Conference_nonomitting	non-omitting	4	stubborn	valid	5	585	195	2170 ms
Conference_nonomitting	non-omitting	4	causal	invalid	9	60359	19781	21488 ms

Figure 5. Experimental Results

GOLOG [16] is a programming language based on first-order language designed for representing dynamically changing systems. A GOLOG program specifies the behavior of the agents in the system. The program is then evaluated with a theorem prover, and thus assertions made in the program can be checked for validity. The problem of inferring inductive invariants in first-order transition systems is more challenging that of invariant checking, and has received less attention in the context of first-order transitions systems. [18] considers precisely this problem, for certain classes of transition systems for which the transition relation is given by formulas in the $\exists^*\forall^*$FOL fragment together with a background theory. The authors show, in particular, that inferring universal inductive invariants is decidable when the transition relation is expressed by formulas with unary predicates and a single binary predicate restricted by the background theory of singly-linked-lists. The same problem becomes undecidable when the binary symbol is not restricted by a background theory. In our work, the termination argument relies on also imposing certain constraints on the structure of the transition system (in particular, workflows are leveled), rather than on the formulas alone.

Business processes are another type of multi-agent workflow systems in which agents perform activities in a predefined flow. In contrast to the workflows considered here, where workflow steps are executed synchronously by a set of agents, in business processes activities are executed asynchronously. Information flow in business processes has been considered, e.g., in [3], which uses the MASK framework for possibilistic information flow security [17] to manually prove the absence of information leaks. Concrete workflow systems, such as conference management systems [15], [1], or a social media platform [4], have recently been proposed and analyzed. In contrast to these works, which focus on the verification of one specific system, we propose a verification approach for arbitrary workflows.

X. Conclusion

The goal of this paper was to provide methods for verifying complex NDA properties in practically relevant workflows. We proceeded in two steps. First, we simplified NDA by encoding execution of two traces together with both the agent model and declassification into the workflow. For verifying the simplified property, we then relied on inductive universal invariants of the resulting workflows. For checking inductiveness as well as for inferring inductive invariants, we found it useful to *abstract* arbitrary formulas by universal formulas. We also applied a complete method for second-order quantifier elimination. For a non-trivial class of workflows, we thus succeeded to compute the *best*, i.e., weakest universal invariant which is inductive. We practically evaluated these methods on example workflows, which formalize non-trivial aspects of conference management systems. On these examples, our algorithms turned out to be surprisingly fast. It remains open whether one can extend the class of workflows for which best inductive universal invariants can be inferred. Also, more experimentation is required to better evaluate how well the proposed methods work in practice. It would also be interesting to search for further agent models possibly occurring in practice.

Acknowledgments.: 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” and in the doctorate program “Program and Model Analysis - PUMA” (no. 1480).

References

[1] Arapinis, M., Bursuc, S., Ryan, M.: Privacy supporting cloud computing: Confichair, a case study. In: Proc. POST 2012. pp. 89–108. Springer Verlag (2012)

[2] Ball, T., Bjørner, N., Gember, A., Itzhaky, S., Karbyshev, A., Sagiv, M., Schapira, M., Valadarsky, A.: Vericon: towards verifying controller programs in software-defined networks. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2014). pp. 282–293. ACM (2014)

[3] Bauereiß, T., Hutter, D.: Information flow control for workflow management systems. it - Information Technology 56(6), 294–299 (2014)

[4] Bauereiß, T., Pesenti Gritti, A., Popescu, A., Raimondi, F.: Cosmedis: A distributed social media platform with formally verified confidentiality guarantees. In: IEEE Symposium on Security and Privacy, SP 2017. pp. 729–748. IEEE Computer Society (2017)

[5] Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives in Mathematical Logic, Springer (1997)

[6] Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security 18(6), 1157–1210 (2010)

[7] Cousot, P., Cousot, R., Mauborgne, L.: Logical abstract domains and interpretations. In: The Future of Software Engineering. pp. 48–71. Springer (2011)

[8] De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)

[9] Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press, San Diego, CA, USA (1972)

[10] Finkbeiner, B., Müller, C., Seidl, H., Zălinescu, E.: Verifying security policies in multi-agent workflows with loops. In: 15th ACM Conf. on Computer and Communications Security (CCS'17). pp. 633–645. ACM Press (2017)

[11] Finkbeiner, B., Seidl, H., Müller, C.: Specifying and verifying secrecy in workflows with arbitrarily many agents. In: Proc. of the 14th Int. Symposium on Automated Technology for Verification and Analysis (ATVA 2016). Lecture Notes in Computer Science, vol. 9938, pp. 157–173 (2016)

[12] Gabbay, D.M., Ohlbach, H.J.: Quantifier elimination in second-order predicate logic. In: Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'92). pp. 425–435. Morgan Kaufmann (1992)

[13] Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proc. of the IEEE Symposium on Security and Privacy. pp. 11–20 (1982)

[14] Goranko, V., Hustadt, U., Schmidt, R.A., Vakarelov, D.: SCAN is complete for all sahlqvist formulae. In: Relational and Kleene-Algebraic Methods in Computer Science: 7th Int. Seminar on Relational Methods in Computer Science and 2nd Int. Workshop on Applications of Kleene Algebra. Lecture Notes in Computer Science, vol. 3051, pp. 149–162. Springer (2004)

[15] Kanav, S., Lammich, P., Popescu, A.: A conference management system with verified document confidentiality. In: Proc. of the 26th Int. Conf. on Computer Aided Verification (CAV 2014). pp. 167–183. Springer Verlag (2014)

[16] Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: Golog: A logic programming language for dynamic domains. The Journal of Logic Programming 31(1), 59 – 83 (1997)

[17] Mantel, H.: Possibilistic Definitions of Security – An Assembly Kit. In: Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW). pp. 185–199. IEEE Computer Society (July 3–5 2000)

[18] Padon, O., Immerman, N., Shoham, S., Karbyshev, A., Sagiv, M.: Decidability of inferring inductive invariants. In: Proc. of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016. pp. 217–231. ACM (2016)

[19] Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: decidable reasoning about distributed protocols. PACMPL 1(OOPSLA), 108:1–108:31 (2017)

[20] Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: safety verification by interactive generalization. In: Proc. of the 37th ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI 2016. pp. 614–630 (2016)

Appendix A
Transformation for Oracles

In Section I, we introduced the restriction that oracles are only queried in non-**may** blocks, i.e., blocks of the form **forall** $\bar{x}$. *stmts* which additionally satisfy:

1) Every statement in *stmts* makes use of the same oracle;
2) Each guard querying an oracle O is of the form $\varphi' \wedge O\bar{x}$, i.e., the sequence of arguments of O coincides with the block's quantified variables.

Here, we will show how to transform any given workflow to adhere to this restriction.

Intuitively, the idea is to introduce for each oracle O, an auxiliary predicate R_O in $\mathcal{R}_{wf}$ in which the relation of O is stored for use in the next block. The predicate R_O can then be set before its use by means of a single non-**may** block comprising of a single statement, and reset to the empty relation afterwards. The problem with this construction is that, according to our definition of *observable* tuples, each tuple $(a_1, \ldots, a_k)$ of the oracle would become known to the agent a_1. If this flow of information is undesirable, it can be avoided by introducing an extra first component to the auxiliary predicate R_O and fill it with some constant c.

The most natural way to exclude this constant from the agents of the system, is to introduce *sorts* into the model as well as into the logic (see [10] for details). Then the constant could simply be assigned a non-agent sort.

Alternatively, the declassification condition could be used to explicitly allow flows of information to this constant and fix the agent model for c to *stubborn*.

Example 5. *Assume we are given a workflow that contains the block:*

$$\begin{array}{l} \textbf{forall}\ x, y, z\ \textbf{may}. \\ \quad R(x,z) \vee \neg O_1(x) \to S \mathrel{+}= (x,z) \\ \quad O_2(y,z) \to T \mathrel{+}= (x,y,z) \end{array}$$

It will be transformed to the following sequence of blocks:

$$\begin{array}{l} \textbf{forall}\ x.\ O_1(x) \to R_{O_1} \mathrel{+}= (c,x) \\ \textbf{forall}\ y,z.\ O_2(y,z) \to R_{O_2} \mathrel{+}= (c,y,z) \\ \textbf{forall}\ x,y,z\ \textbf{may}. \\ \quad R(x,z) \vee \neg R_{O_1}(c,x) \to S \mathrel{+}= (x,z) \\ \quad R_{O_2}(c,y,z) \to T \mathrel{+}= (x,y,z) \\ \textbf{forall}\ x.\ \mathit{true} \to R_{O_1} \mathrel{-}= (c,x) \\ \textbf{forall}\ y,z.\ \mathit{true} \to R_{O_2} \mathrel{-}= (c,y,z) \end{array}$$

where the declassification predicates $\mathit{declass}_{O_i}(a)$ are updated to the extended predicates

$$\mathit{declass}_{O_i}(a) \vee (a = c)$$

This transformation only adds non-omitting blocks. So if the initial workflow was non-omitting, the transformed workflow will be non-omitting as well. The same holds for edge-uniformity as well as leveledness — all R_O are fresh and information only flows from O to R_O. However, uniformity is not preserved, since tuples are both added to and subtracted from R_O.

Appendix B
Transformation for Stubborn Agents

For completeness, we also detail the transformation $\mathcal{T}_a^{(s)}$. It transforms the given workflow to compute all pairs of traces that satisfy declassification where all participating agents act *stubbornly*, i.e. according to $\mathit{agent\_model}^{(s)}$.

Given a workflow w (via its CFG) and an agent variable a, we construct a new workflow $\mathcal{T}_a^{(s)}w$ as follows. Let $\bar{\mathcal{R}} = \mathcal{R} \cup \mathcal{R}'$ denote the set of predicates R used by w extended by the corresponding distinct primed predicates R'. For a first-order formula φ with predicates from $\mathcal{R}$, let $[\varphi]'$ denote the formula obtained from φ by replacing each predicate $R \in \mathcal{R}$ with the corresponding predicate R' in $\mathcal{R}'$. Then each edge (u, β, v) of w gives rise to one edge $(u, \mathcal{T}_a^{(s)}\beta, v)$ in $\mathcal{T}_a^{(s)}w$ where the transformation $\mathcal{T}_a^{(s)}$ on sequences of blocks is the concatenation of the transformations of the individual blocks b in β. It is defined as follows.

Case 1. b does not contain a query to an oracle. In this case, $\mathcal{T}_a^{(s)}b$ is obtained from b by adding to every update $\varphi \to R \mathrel{+}= \bar{z}$, the update $[\varphi]' \to R' \mathrel{+}= \bar{z}$, and likewise, for every update $\varphi \to R \mathrel{-}= \bar{z}$, the update $[\varphi]' \to R' \mathrel{-}= \bar{z}$. As a result, the same predicate C_b is used for both updates (in case of **may**) and stubbornness is enforced.

Case 2. b is of the form **forall** $\bar{x}.\,$*stmts* and contains a query to the oracle O. W.l.o.g., we may assume that *each* statement in *stmts* queries O. In order to simulate declassification, we set $\mathcal{T}_a^{(s)}b = b_1 b_2$ for **may** blocks b_1, b_2 where the predicates C_{b_1}, C_{b_2} serve as the possibly distinct versions of the oracle on the two simulated traces. The block b_1 equals **forall** $\bar{x}.$ **may** *stmts*$_1$ where the sequence *stmts*$_1$ is obtained from *stmts* by collecting for every update $\varphi \wedge O\bar{x} \to R \mathrel{+}= \bar{y}$ in *stmts*, the two updates:

$$\begin{array}{l} \varphi \to R \mathrel{+}= \bar{y} \\ [\varphi]' \wedge \mathit{declass}_O(a, \bar{x}) \to R' \mathrel{+}= \bar{y} \end{array}$$

Updates that query the oracle, but remove tuples, are treated analogously. We note that the parameter a of the declassification formula *declass*, is considered as a *constant* in the workflow $\mathcal{T}_a^{(s)}w$.

Moreover, b_2 equals **forall** $\bar{x}.$ **may** *stmts*$_2$ where the sequence *stmts*$_2$ is obtained from *stmts* by collecting for every update $\varphi \wedge O(\bar{x}) \to R \mathrel{+}= \bar{y}$ in *stmts*, the update:

$$[\varphi]' \wedge \neg\mathit{declass}_O(a, \bar{x}) \to R' \mathrel{+}= \bar{y}$$

Again, updates that query the oracle, but remove tuples, are treated analogously.

Example 6. *Consider the block*

forall $x, p, r.\ \mathit{Assign}(x,p) \wedge O(x,p,r) \to \mathit{Review} \mathrel{+}= (x,p,r)$

Then the transformation results in the following two blocks:

$$\begin{array}{l}
\textbf{forall}\ x,p,r\ \textbf{may}. \\
\quad \mathit{Assign}(x,p) \to \mathit{Review} \mathrel{+}= (x,p,r) \\
\quad \mathit{Assign}'(x,p) \wedge (\neg\mathit{Conf}(a,p) \vee \neg\mathit{Conf}'(a,p)) \\
\qquad\qquad\qquad\qquad\qquad \to \mathit{Review}' \mathrel{+}= (x,p,r) \\
\textbf{forall}\ x,p,r\ \textbf{may}. \\
\quad \mathit{Assign}'(x,p) \wedge (\mathit{Conf}(a,p) \wedge \mathit{Conf}'(a,p)) \\
\qquad\qquad\qquad\qquad\qquad \to \mathit{Review}' \mathrel{+}= (x,p,r) \quad \square
\end{array}$$

The workflow $\mathcal{T}_a^{(s)}w$ captures all pairs of traces of w that satisfy the assumptions of stubbornness and declassification.

Appendix C
Proof of Theorem 7

Theorem 7. *Let m be the maximal number of distinct substitutions in Θ. Furthermore, let r denote the number of levels of predicates in $\mathcal{R}_{wf}$. Assume that ψ is a first-order formula with predicates from $\mathcal{R}_{wf}$. Assume that $\theta_\beta = \theta_h \circ \ldots \circ \theta_1$ is a sequence of substitutions corresponding to a sequence of blocks β of length h. Then there is a sub-sequence $\theta' = \theta_{j_1} \circ \ldots \circ \theta_{j_l}$ of θ_β with $l < (1 + \frac{m}{r})^r$ such that the following equivalence holds.*

$$\forall A_1 \ldots A_s.\ (\theta_\beta \psi)^\sharp \quad\equiv\quad \forall A_{j_1} \ldots A_{j_l}.\ (\theta' \psi)^\sharp \tag{8}$$

Proof. The proof is based on the observation that the substitution corresponding to the same block b may produce only finitely often new contributions to the composition — at least, if the corresponding second-order quantifiers are later eliminated.

Assume that θ' is a subsequence of θ of minimal length so that (8) holds. W.l.o.g., we assume that each predicate can be assigned a distinct *level* $1 < \ldots < r$ so that for each block b, all updated predicates in that block receive the same level. Let $m_i \geq 1$ denote the number of substitutions in Θ of the predicates with level i, and $B[i]$ the maximal number of occurrences of substitutions θ_{j_λ} in θ' substituting predicates of level at most i.

We claim that for each $i = 0, \ldots, r$, $B[i] \leq (m_1 + 1) \cdot \ldots \cdot (m_i + 1) - 1$ holds. In particular, $B[0] = 0$.

Consider a level $i \geq 1$. First we claim that θ' does not contain two occurrences of substitutions corresponding to the same block of level i so that inbetween there are only substitutions corresponding to blocks of the same level or higher.

Assume for a contradiction, this were not the case, i.e., there are $j_\lambda < j_{\lambda'}$ so that θ_{j_λ} and $\theta_{j_{\lambda'}}$ both are obtained from θ_b for some block b. Let θ'' denote the substitution where $\theta_{j_{\lambda'}}$ has been removed. Let ψ' denote the negation normal form of the formula which is obtained by applying the substitution θ' to ψ, and ψ'' the negation normal form of $\theta''\psi$. Then ψ' is obtained from ψ'' by replacing some positive literals $A_{j_\lambda}\bar{z}$ with the disjunction $A_{j_\lambda}\bar{z} \vee A_{j_{\lambda'}}\bar{z}$, and some negative literals $\neg A_{j_\lambda}\bar{z}$ with the conjunction $\neg A_{j_\lambda}\bar{z} \wedge \neg A_{j_{\lambda'}}\bar{z}$. The same holds true after abstraction of existentials and bringing into prenex normal form. In particular, both formulas result in the same quantifier prefix. Now consider the corresponding conjunctive normal forms $\forall x. \bigwedge S'$ and $\forall x. \bigwedge S''$ of the resulting formulas (where S' and S'' are the corresponding sets of clauses) which are obtained by exhaustively applying the distributivity laws. Then the following properties hold:

1) If $A_{j_{\lambda'}}\bar{z}$ occurs in a clause of S', then also $A_{j_\lambda}\bar{z}$; and
2) If $c' \vee \neg A_{j_{\lambda'}}\bar{z}$ is a clause of S', then also $c' \vee \neg A_{j_\lambda}\bar{z}$; and
3) S'' is obtained from S' by dropping all positive occurrences of the predicate $A_{j_{\lambda'}}\bar{z}$ and by removing all clauses containing negative occurrences of $A_{j_{\lambda'}}\bar{z}$.

Thus, A_{j_λ} dominates $A_{j_{\lambda'}}$ in the conjunctive normal form of $(\theta'\psi)^\sharp$. From that it follows that $\theta_{j_{\lambda'}}$ can be removed without changing the equivalence (8) — which leads to a contradiction.

Now assume that $B[i-1]$ is the maximal length of the subsequence of substitutions of the $i-1$ least predicates in θ', and b is a block at level i. Then a substitution corresponding to b can occur at most $B[i-1]+1$ times. We conclude that the maximal number $B[i]$ of substitutions of at level at most i can be bounded by

$$\begin{array}{rcl}
B[i] & \leq & (B[i-1]+1) \cdot m_i + B[i-1] \\
 & < & (B[i-1]+1) \cdot (m_i + 1) \\
 & = & (m_1 + 1) \cdot \ldots \cdot (m_{i-1} + 1) \cdot (m_i + 1)
\end{array}$$

— what we wanted to prove. $\square$