Inductive Invariants for Noninterference in Multi-Agent Workflows

CSF 2018, Oxford, UK

Christian Müller, Helmut Seidl, Eugen Zălinescu

July 11, 2018

Section 1

Setting

Motivation

Conference Management (Easychair, HotCRP, . . .)
Used by authors, reviewers, PC chair, . . .

Motivation

Conference Management (Easychair, HotCRP, . . .)
Used by authors, reviewers, PC chair, . . .

HotCRP.com

2.60 19.Jul.2013
Major new feature: Paper managers. Administrators can assign PC members to "manage" individual papers. These PC members gain admin rights over those papers, and can, for example, assign reviewers as usual.
This required extensive rearchitecting of system internals to (hopefully) avoid information leaks.

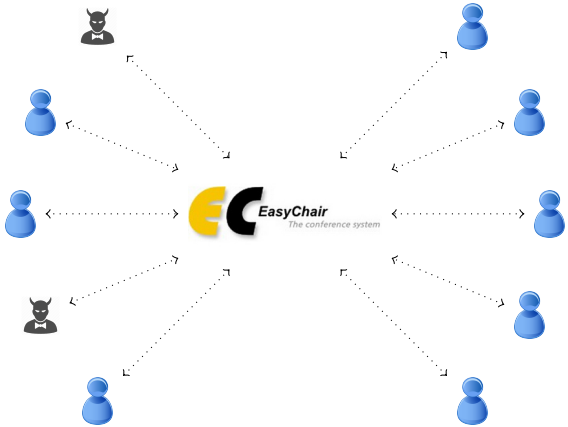
Motivation

Conference Management (Easychair, HotCRP, . . .)
Used by authors, reviewers, PC chair, . . .

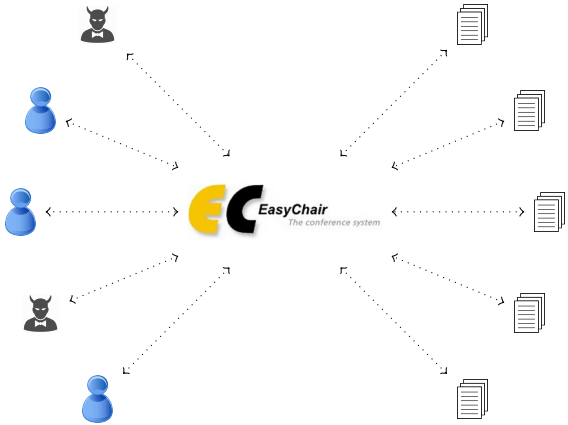
2.60 19.Jul.2013
Major new feature: Paper managers. Administrators can assign PC members to "manage" individual papers. These PC members gain admin rights over those papers, and can, for example, assign reviewers as usual.
This required extensive rearchitecting of system internals to (hopefully) avoid information leaks.

Systems need to protect sensitive information!

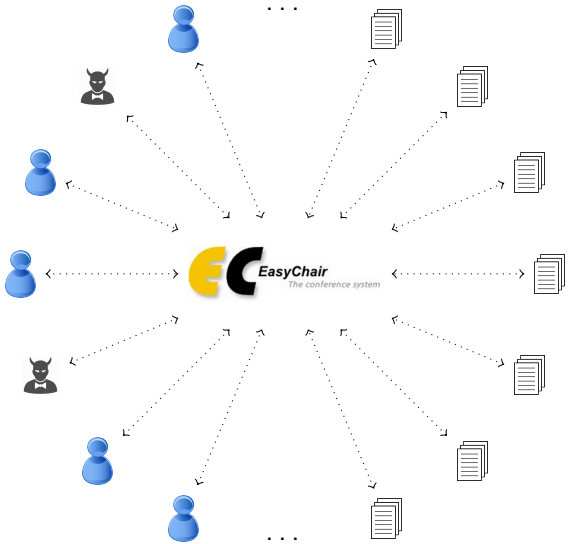
Challenges



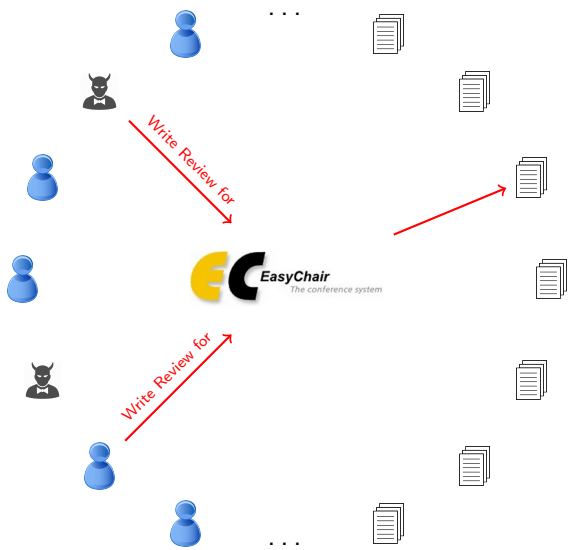
Challenges



Challenges



Challenges


Write Review for
EasyChair
The conference system
Write Review for

Conflict Declaration

PC Members may declare conflicts of interest with papers.

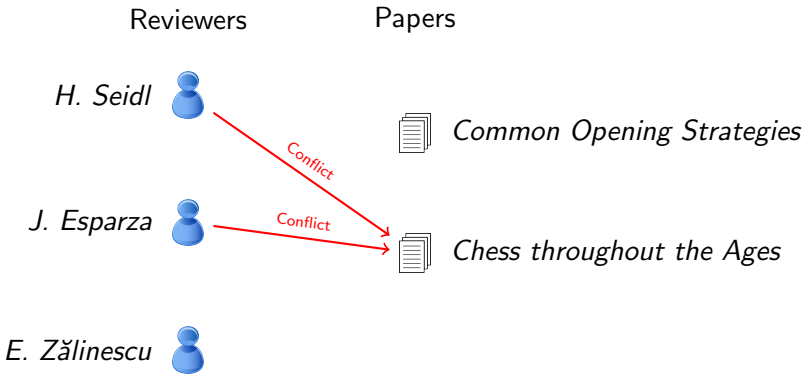
Conflict Declaration

PC Members may declare conflicts of interest with papers.

Reviewers	Papers
H. Seidl	*Common Opening Strategies*
J. Esparza	*Chess throughout the Ages*
E. Zălinescu	

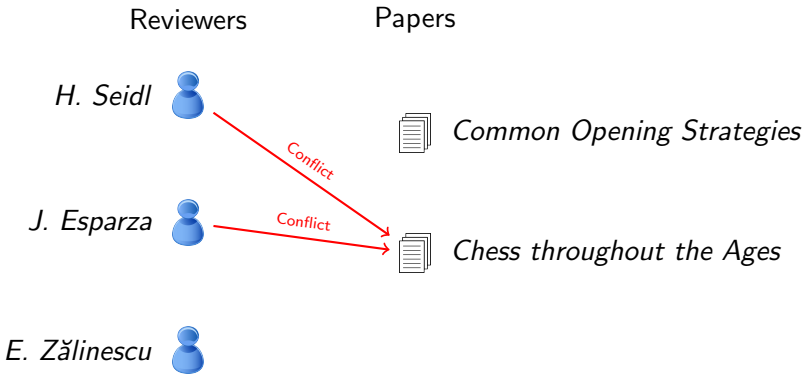
Conflict Declaration

PC Members may declare conflicts of interest with papers.



Conflict Declaration

PC Members may declare conflicts of interest with papers.



forall x: *PCMember*, p: *Paper*. **may** *true* $\rightarrow$ *Conflict* $+= (x, p)$

Complete Example Workflow

% 1. PC members may declare conflicts

% 2. PC members are assigned to papers

% 3. PC members write reviews for papers

% 4. PC members read all reviews

% 5. PC members can update their reviews

Complete Example Workflow

% 1. PC members may declare conflicts
forall x: *PCMember*, p: *Paper* **may**. *true* $\rightarrow$ *Conflict* $+=$ (x, p)

% 2. PC members are assigned to papers
forall x: *PCMember*, p: *Paper* **may**. $\neg$*Conflict*(x, p) $\rightarrow$ *Assign* $+=$ (x, p)

% 3. PC members write reviews for papers
forall x: *PCMember*, p: *Paper*, r: *Review*.
Assign(x, p) $\wedge$ *Oracle*(x, p, r) $\rightarrow$ *Review* $+=$ (x, p, r)

loop $\{$

% 4. PC members read all reviews
forall x: *PCMember*, y: *PCMember*, p: *Paper*, r: *Review*. **may**
Assign(x, p) $\wedge$ *Review*(y, p, r) $\rightarrow$ *Read* $+=$ (x, p, r)

% 5. PC members can update their reviews
forall x: *PCMember*, p: *Paper*, r: *Review* **may**.
Assign(x, p) $\wedge$ *Oracle*(x, p, r) $\rightarrow$ *Review* $+=$ (x, p, r)

$\}$

Semantics

Semantics of the Workflow in *Sorted First Order Logic*

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

Semantics

Semantics of the Workflow in *Sorted First Order Logic*

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

is expressed by

$$\forall x\colon PCMember, p\colon Paper, r\colon Review.\ Read'(x,p,r) \leftrightarrow$$
$$Read(x,p,r) \vee (\exists y\colon PCMember.\ \ Choice(x,y,p,r) \wedge Assign(x,p) \wedge Review(y,p,r))$$

Semantics

Semantics of the Workflow in *Sorted First Order Logic*

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

is expressed by

$$\forall x\colon PCMember, p\colon Paper, r\colon Review.\ Read'(x,p,r) \leftrightarrow$$
$$Read(x,p,r) \vee (\exists y\colon PCMember.\ \ Choice(x,y,p,r) \wedge Assign(x,p) \wedge Review(y,p,r))$$

Semantics

Semantics of the Workflow in *Sorted First Order Logic*

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

is expressed by

$$\forall x\colon PCMember, p\colon Paper, r\colon Review.\ Read'(x,p,r) \leftrightarrow$$
$$Read(x,p,r) \vee (\exists y\colon PCMember.\ Choice(x,y,p,r) \wedge Assign(x,p) \wedge Review(y,p,r))$$

Semantics

Semantics of the Workflow in *Sorted First Order Logic*

forall x: *PCMember*, y: *PCMember*, p: *Paper*, r: *Review*. **may**

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

is expressed by

$$\forall x\text{: } \mathit{PCMember}, p\text{: } \mathit{Paper}, r\text{: } \mathit{Review}.\ \mathit{Read}'(x, p, r) \leftrightarrow$$

$$\mathit{Read}(x, p, r) \vee (\exists y\text{: } \mathit{PCMember}.\ \ \mathit{Choice}(x, y, p, r) \wedge \mathit{Assign}(x, p) \wedge \mathit{Review}(y, p, r))$$

Noninterference

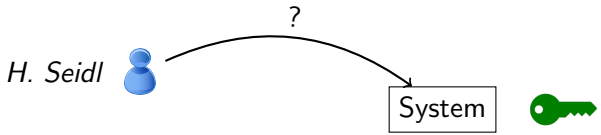
Can any agent learn secret information?

H. Seidl 

System 

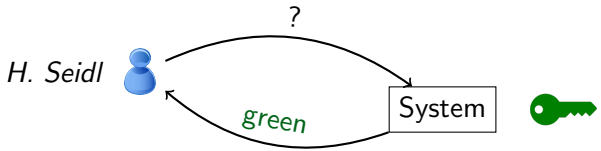
Noninterference

Can any agent learn secret information?



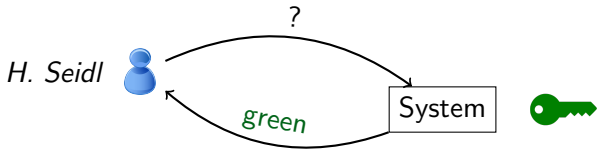
Noninterference

Can any agent learn secret information?



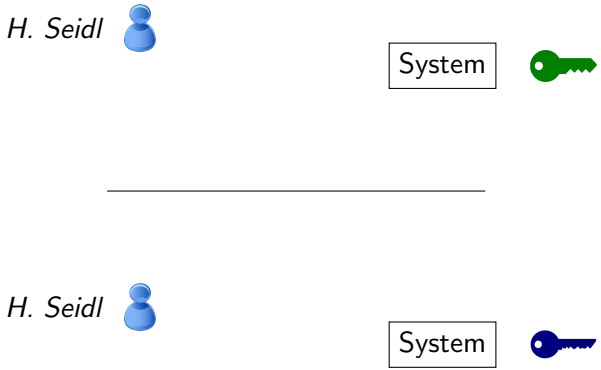
Noninterference

Can any agent learn secret information?

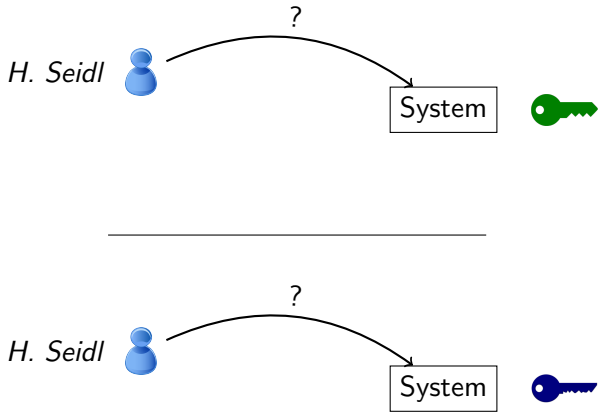


Does *H. Seidl* learn something?

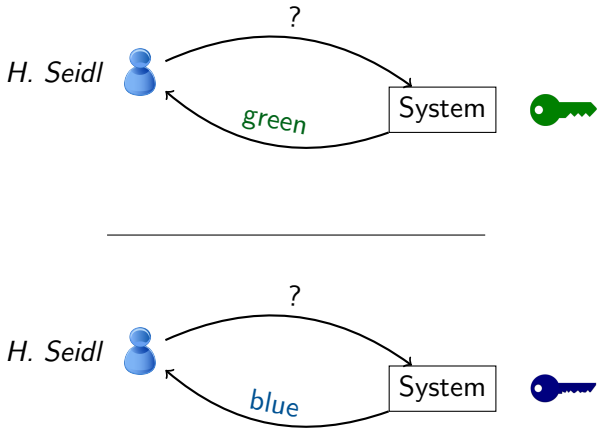
Noninterference — Hyperproperty



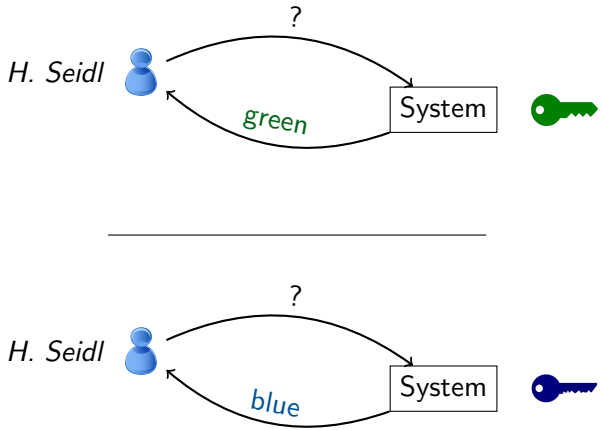
Noninterference — Hyperproperty



Noninterference — Hyperproperty

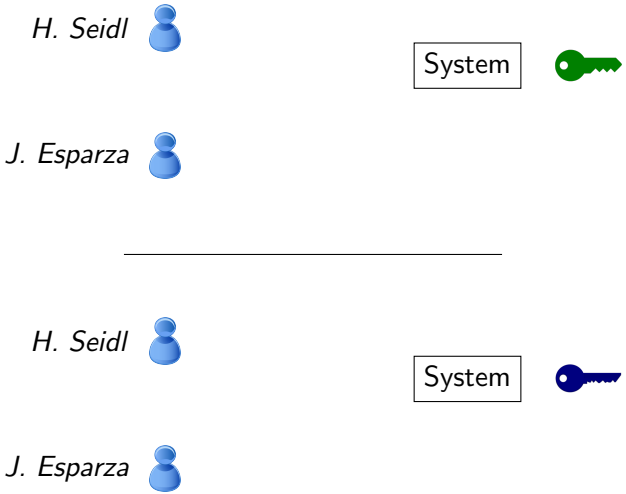


Noninterference — Hyperproperty

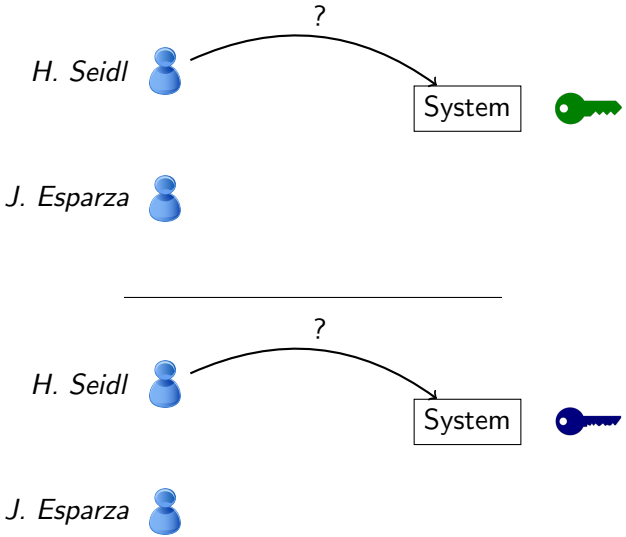


Is this enough?

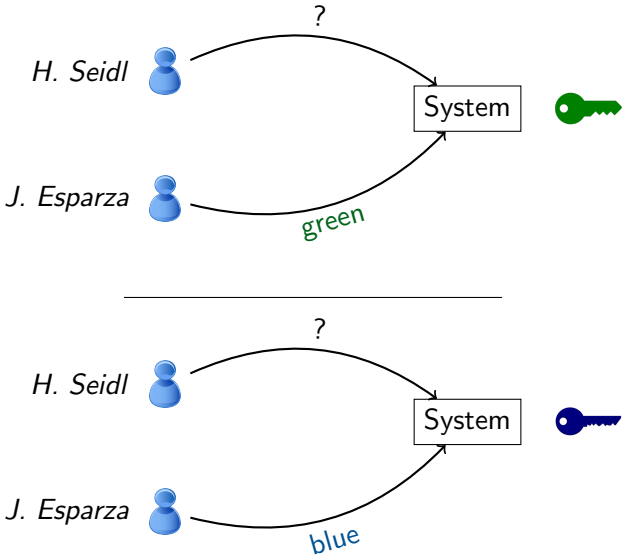
Noninterference



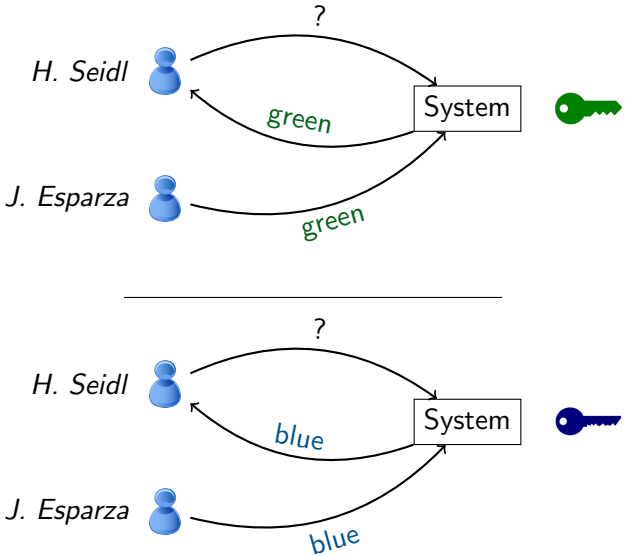
Noninterference



Noninterference



Noninterference



Agent Models

Stubborn Agents

Behavior does not depend on secret information.

Agent Models

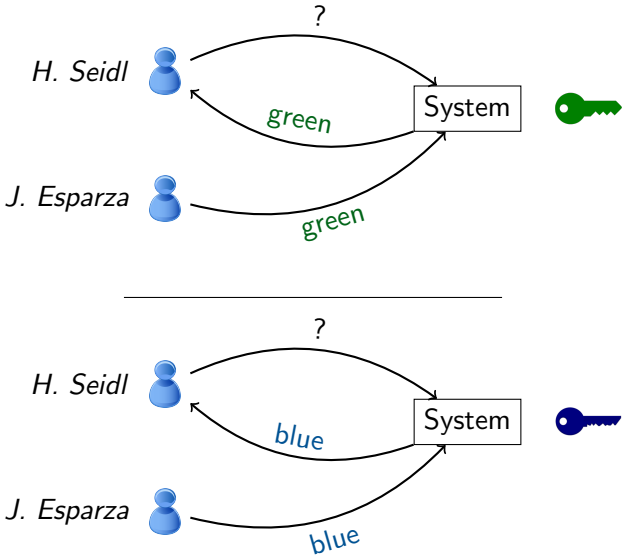
Stubborn Agents

Behavior does not depend on secret information.

Causal Agents

May change their behavior to leak information.
$\Rightarrow$ Causal agents may conspire against the system.

Noninterference



State of the Art

Restricted Classes

Verification problem solved for restricted classes: Translation to $\exists^*$FOLTL[1].

General Case

Verification of Noninterference for general workflows is undecidable[1].

[1] Bernd Finkbeiner et al. "Verifying Security Policies in Multi-agent Workflows with Loops". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017, pp. 633–645. DOI: 10.1145/3133956.3134080. URL: http://doi.acm.org/10.1145/3133956.3134080, p. 1.

Simplifying the desired properties

Noninterference property consists of

Two copies of all relations
Agent model
Declassification
No observable differences

Example: Declassification

$\neg \mathit{Conflict}_1(x, p) \vee \neg \mathit{Conflict}_2(x, p)$

Example: No observable differences

$\mathbf{G} \bigwedge \forall \bar{z}.\ R_1(a, \bar{z}) \leftrightarrow R_2(a, \bar{z})$

Simplifying the desired properties

Noninterference property consists of

Two copies of all relations	Can be encoded into workflow
Agent model	Can be encoded into workflow
Declassification	Can be encoded into workflow
No observable differences	Universal **G** property

Example: Declassification

$\neg Conflict_1(x, p) \vee \neg Conflict_2(x, p)$

Example: No observable differences

$\mathbf{G} \bigwedge \forall \bar{z}.\ R_1(a, \bar{z}) \leftrightarrow R_2(a, \bar{z})$

Invariants

Solution

Absence of Noninterference violations can be encoded into an invariant.

$\Rightarrow$ Prove workflows *safe* using invariants

Section 2

Proving Invariants

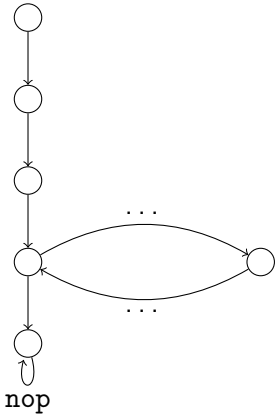
Invariants

forall $x, p.\ true \rightarrow Conflict \mathrel{+}= (x, p)$

forall $x, p.\ \neg Conflict(x, p) \rightarrow Assign \mathrel{+}= (x, p)$

forall $x, p, r.$
$Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$

loop $\{ \ldots \}$



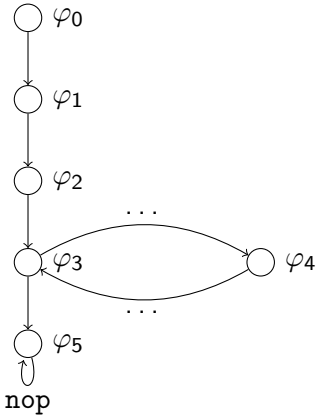
Invariants

forall $x, p.\ \mathit{true} \rightarrow \mathit{Conflict} \mathrel{+}= (x, p)$

forall $x, p.\ \neg \mathit{Conflict}(x, p) \rightarrow \mathit{Assign} \mathrel{+}= (x, p)$

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

loop $\{ \ldots \}$



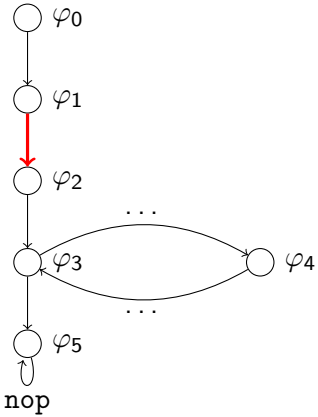
Invariants

forall $x, p.\ \mathit{true} \rightarrow \mathit{Conflict} \mathrel{+}= (x, p)$

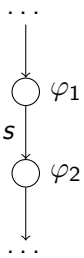
forall $x, p.\ \neg \mathit{Conflict}(x, p) \rightarrow \mathit{Assign} \mathrel{+}= (x, p)$

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

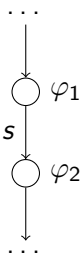
loop $\{ \ldots \}$



Proving Invariants inductive

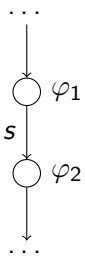


Proving Invariants inductive



Does the statement respect the invariant?

Proving Invariants inductive

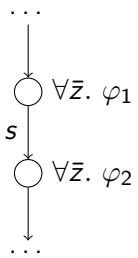


Does the statement respect the invariant?

Check if

$$\varphi_1 \rightarrow \mathsf{WP}[\![s]\!](\varphi_2)$$

Proving Invariants inductive

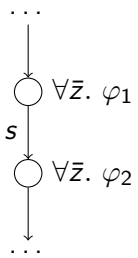


Does the statement respect the invariant?

Check if

$$\forall \bar{z}.\ \varphi_1 \rightarrow \mathsf{WP}[\![s]\!](\forall \bar{z}.\ \varphi_2)$$

Proving Invariants inductive



Does the statement respect the invariant?

Check if

$$\forall \bar{z}.\ \varphi_1 \rightarrow \mathsf{WP}[\![s]\!](\forall \bar{z}.\ \varphi_2) \qquad \checkmark$$

Proving Invariants inductive

How to check if an invariant holds

For an invariant φ

1. Check inductiveness for all edges of the CFG (φ is *inductive*)
2. Check if the initial state implies φ_0 (φ is *safe*)

Proving Invariants inductive

How to check if an invariant holds

For an invariant φ

1. Check inductiveness for all edges of the CFG (φ is *inductive*)
2. Check if the initial state implies φ_0 (φ is *safe*)

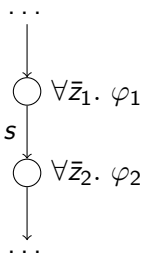
Theorem (Correctness)

For a given workflow w with and a universal invariant φ, it is decidable if φ is inductive and safe for w.

Section 3

Inferring Inductive Invariants

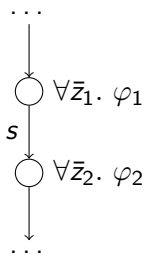
Inferring Invariants



Can we strengthen if it does not?

$$\forall\ \bar{z}_1.\varphi_1 \rightarrow \mathsf{WP}[\![s]\!](\forall\ \bar{z}_2.\varphi_2) \qquad \text{✗}$$

Inferring Invariants



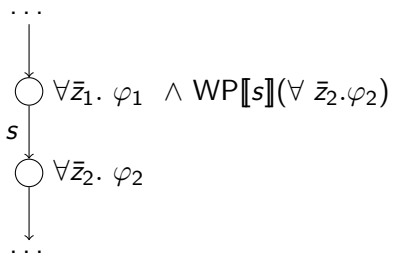
Can we strengthen if it does not?

$$\forall\ \bar{z}_1.\varphi_1 \rightarrow \mathsf{WP}[\![s]\!](\forall\ \bar{z}_2.\varphi_2) \qquad \text{✗}$$

Yes - replace by

$$\forall\ \bar{z}_1.\varphi_1 \wedge \mathsf{WP}[\![s]\!](\forall\ \bar{z}_2.\varphi_2)$$

Inferring Invariants



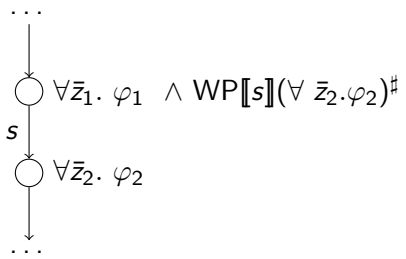
Can we strengthen if it does not?

$$\forall\ \bar{z}_1.\varphi_1 \rightarrow \mathsf{WP}[\![s]\!](\forall\ \bar{z}_2.\varphi_2) \qquad \text{✗}$$

Yes - replace by

$$\forall\ \bar{z}_1.\varphi_1 \wedge \mathsf{WP}[\![s]\!](\forall\ \bar{z}_2.\varphi_2)$$

Inferring Invariants



Can we strengthen if it does not?

$$\forall\ \bar{z}_1.\varphi_1 \rightarrow \mathrm{WP}[\![s]\!](\forall\ \bar{z}_2.\varphi_2) \qquad \text{✗}$$

Yes - replace by

$$\forall\ \bar{z}_1.\varphi_1 \wedge \mathrm{WP}[\![s]\!](\forall\ \bar{z}_2.\varphi_2)^\sharp$$

(and abstract existential quantifiers)

Results

Theorem (Inference)

For a universal invariant φ, if fixpoint iteration terminates, we obtain an inductive universal invariant that implies φ.

Results

Theorem (Inference)

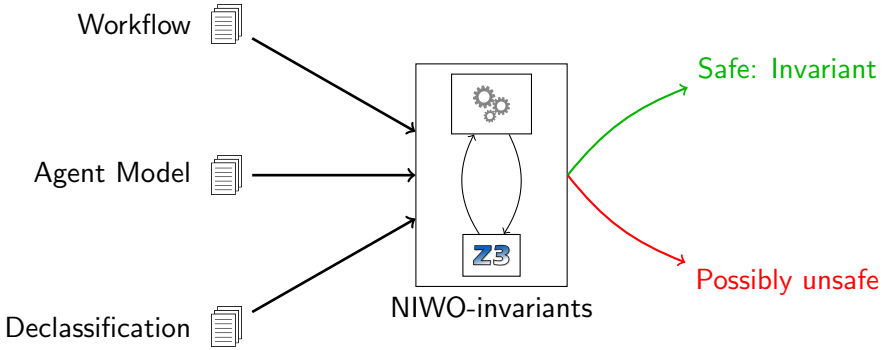
For a universal invariant φ, if fixpoint iteration terminates, we obtain an inductive universal invariant that implies φ.

Corollary (Noninterference)

For a given workflow w, if fixpoint iteration terminates, we obtain an inductive universal invariant that implies Noninterference for w.

Implementation

Meet *NIWO-invariants*, an automatic analyzer for **N**on**i**nterference in **Wo**rkflows



Implementation

Name	Size	Model	Result	Time
Conference_linear	4	stubborn	safe	338 ms
Conference_linear	4	causal	unsafe	1537 ms
Conference_linear_fixed	5	stubborn	safe	322 ms
Conference_linear_fixed	5	causal	safe	2352 ms
Conference_nonomitting	4	stubborn	safe	2170 ms
Conference_nonomitting	4	causal	unsafe	21488 ms
Conference_omitting	6	stubborn	safe	347 ms
Conference_omitting	6	causal	unsafe	2924 ms
Conference_omitting_fixed	7	stubborn	safe	1059 ms
Conference_omitting_fixed	7	causal	safe	5450 ms

Will it terminate?

In general, termination not guaranteed.

Will it terminate?

In general, termination not guaranteed.

Termination Guarantee

1. Syntactic Fragment
2. Upper bound on # causal agents

Contributions

1. Transform Noninterference problem for workflows to invariant checking
2. Introduced invariant verification and inference for general workflows
3. Built *NIWO-invariants*, an inferring invariant solver for Noninterference in general workflows