Multi-Agent Workflows

joint work with Bernd Finkbeiner, Helmut Seidl, Eugen Zălinescu

Christian Müller

August 12, 2019

Overview

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.

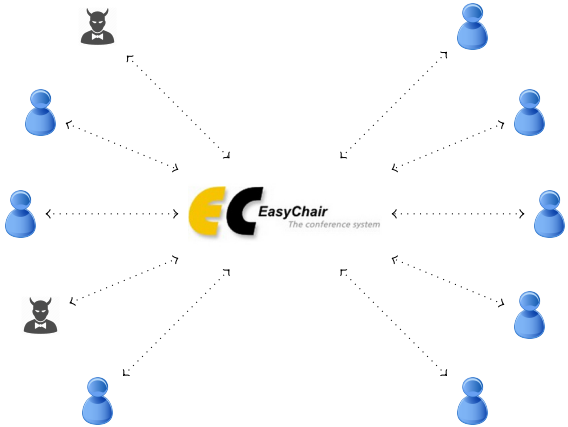
Motivation

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

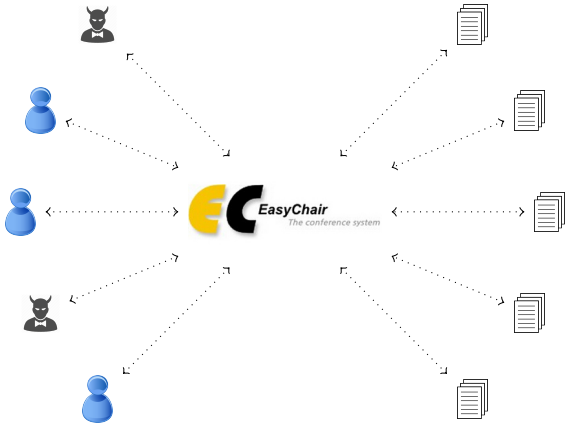
2.60 19.Jul.2013
Major new feature: Paper managers. Administrators can assign PC members to "manage" individual papers. These PC members gain admin rights over those papers, and can, for example, assign reviewers as usual.
This required extensive rearchitecting of system internals to (hopefully) avoid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

Systems need to protect sensitive information!

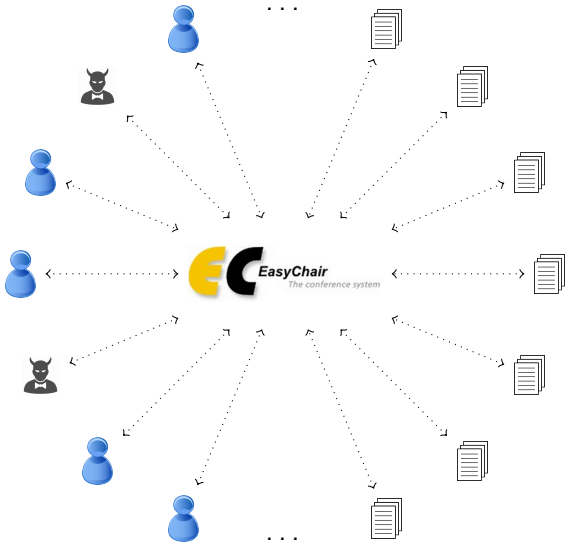
Challenges



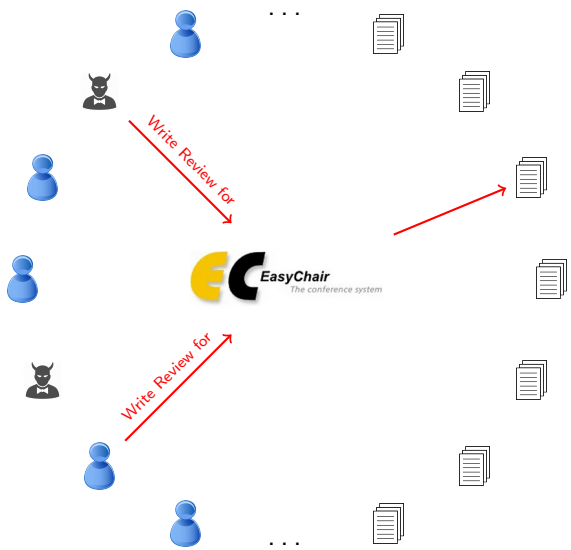
Challenges



Challenges



Challenges


Write Review for
EasyChair
The conference system
Write Review for

Conference Management System

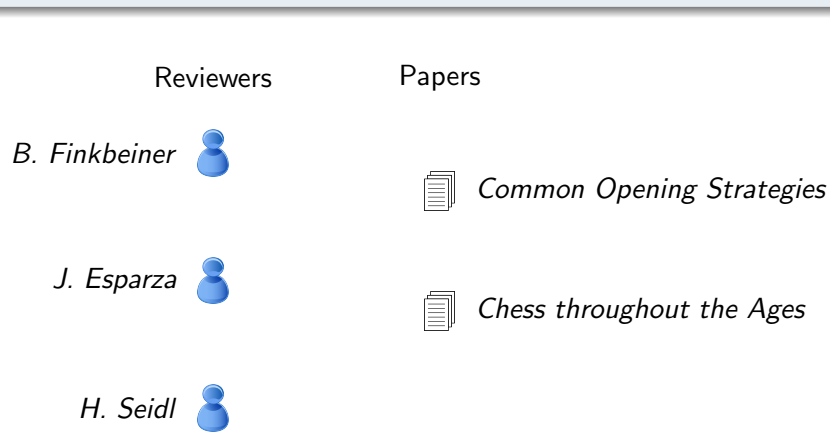
1. Declare Conflicts of Interest
2. Assign Papers to Reviewers
3. Write initial Reviews
4. Loop:
    1. Discuss Reviews
    2. Update Reviews

Conflict Declaration

Reviewers may declare conflicts of interest with papers.

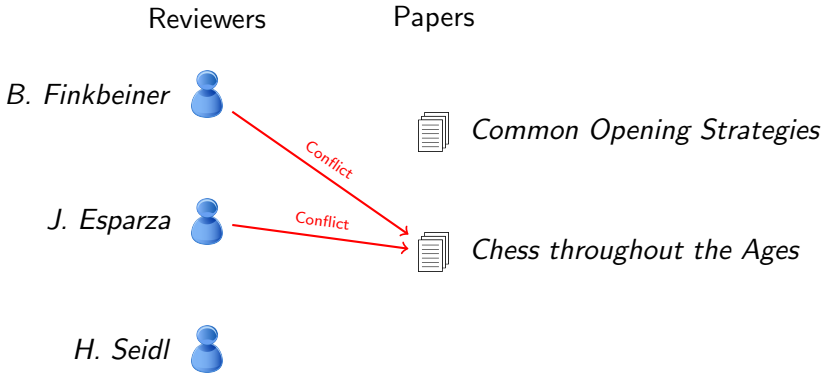
Conflict Declaration

Reviewers may declare conflicts of interest with papers.



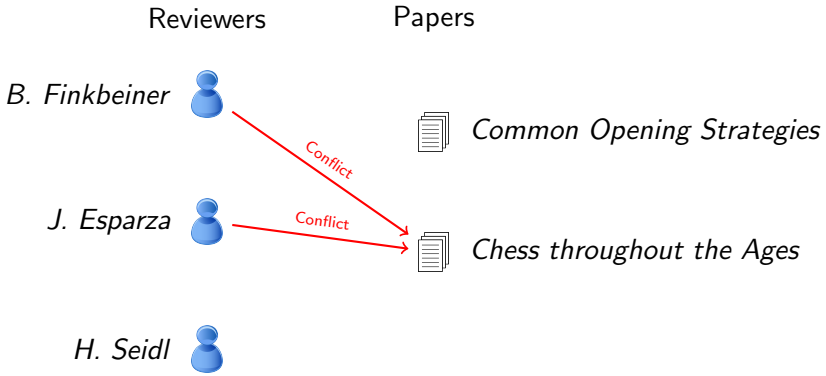
Conflict Declaration

Reviewers may declare conflicts of interest with papers.



Conflict Declaration

Reviewers may declare conflicts of interest with papers.



forall x: PC, p: $Paper$. **may** $true \rightarrow Conflict \mathrel{+}= (x, p)$

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

forall x: PC, p: $Paper$, r: $Review$. **may**

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

Semantics

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

is expressed by

$$\forall x\colon PC, p\colon Paper, r\colon Review.\ \mathbf{X}\ Review(x,p,r) \leftrightarrow$$
$$Review(x,p,r) \vee (Choice(x,p,r) \wedge Assign(x,p) \wedge Oracle(x,p,r))$$

Semantics

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

is expressed by

$$\forall x\colon PC, p\colon Paper, r\colon Review.\ \mathbf{X}\ Review(x,p,r) \leftrightarrow$$
$$Review(x,p,r) \vee (Choice(x,p,r) \wedge Assign(x,p) \wedge Oracle(x,p,r))$$

Semantics

forall $x\colon PC, p\colon Paper, r\colon Review$. **may**

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

is expressed by

$$\forall x\colon PC, p\colon Paper, r\colon Review.\ \mathbf{X}\ Review(x,p,r) \leftrightarrow$$

$$Review(x,p,r) \vee (Choice(x,p,r) \wedge Assign(x,p) \wedge Oracle(x,p,r))$$

Semantics

forall x: PC, p: $Paper$, r: $Review$. **may**

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

is expressed by

$$\forall x\text{: } PC, p\text{: } Paper, r\text{: } Review.\ \mathbf{X}\ Review(x, p, r) \leftrightarrow$$

$$Review(x, p, r) \vee (Choice(x, p, r) \wedge Assign(x, p) \wedge Oracle(x, p, r))$$

Semantics

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

is expressed by

$$\forall x\colon PC, p\colon Paper, r\colon Review.\ \mathbf{X}\, Review(x,p,r) \leftrightarrow$$
$$Review(x,p,r) \vee (Choice(x,p,r) \wedge Assign(x,p) \wedge Oracle(x,p,r))$$

Semantics of the Workflow in *Sorted First Order Temporal Logic (FOLTL)*

Semantics

forall x: PC, p: $Paper$, r: $Review$. **may**

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

is expressed by

$$\forall x\text{: } PC, p\text{: } Paper, r\text{: } Review.\ \mathbf{X}\ Review(x, p, r) \leftrightarrow$$

$$Review(x, p, r) \vee (Choice(x, p, r) \wedge Assign(x, p) \wedge Oracle(x, p, r))$$

Semantics of the Workflow in *Sorted First Order Temporal Logic (FOLTL)*

If all variables reappear, we call the workflow *non-omitting*

Properties

Reachability

Can we reach a state where agent a is assigned paper p?

$$\mathbf{F}\mathit{Assign}(a, p)$$

Properties

Reachability

Can we reach a state where agent a is assigned paper p?

$$\mathbf{F}\mathit{Assign}(a, p)$$

Safety

Can we avoid states where any agent is assigned a conflicting paper?

$$\mathbf{G}\forall a, p.\neg(\mathit{Assign}(a, p) \wedge \mathit{Conflict}(a, p))$$

Properties

Reachability

Can we reach a state where agent a is assigned paper p?

$$\mathbf{F}\mathit{Assign}(a, p)$$

Safety

Can we avoid states where any agent is assigned a conflicting paper?

$$\mathbf{G}\forall a, p.\neg(\mathit{Assign}(a, p) \wedge \mathit{Conflict}(a, p))$$

Security

Can any agent gain information on reviews of conflicting papers?

Section 2

Verification

Verification

Control Flow

Semantics

Specification

Verification

Control Flow

Semantics

Specification

Complete formal specification

Verification

Control Flow

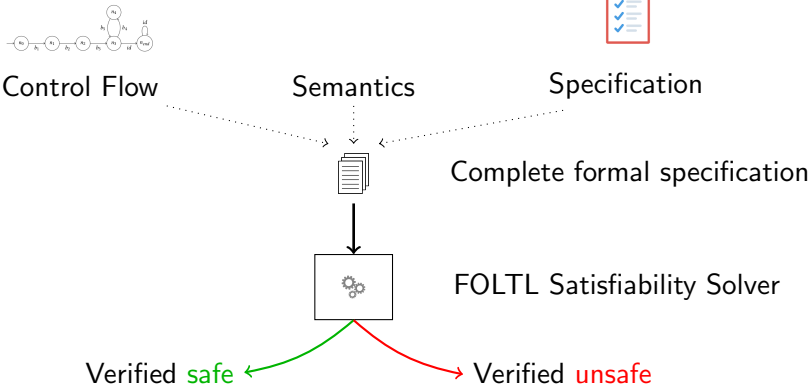
Specification

Semantics

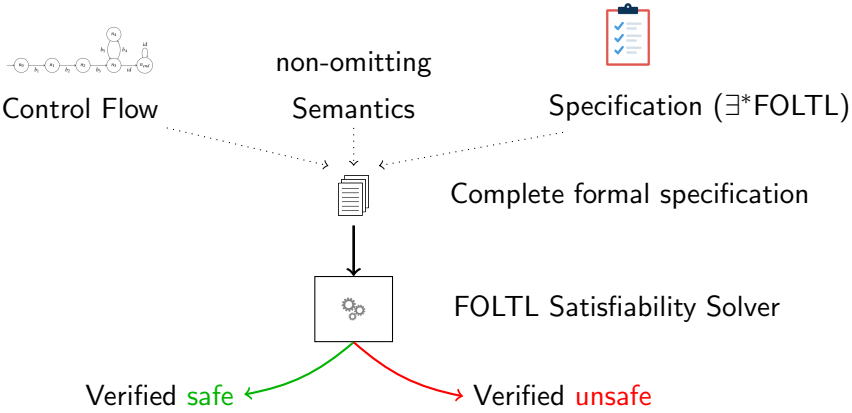
Complete formal specification

FOLTL Satisfiability Solver

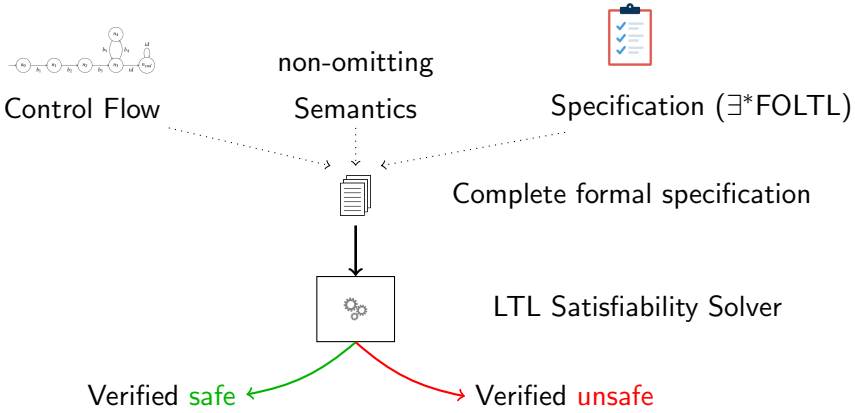
Verification



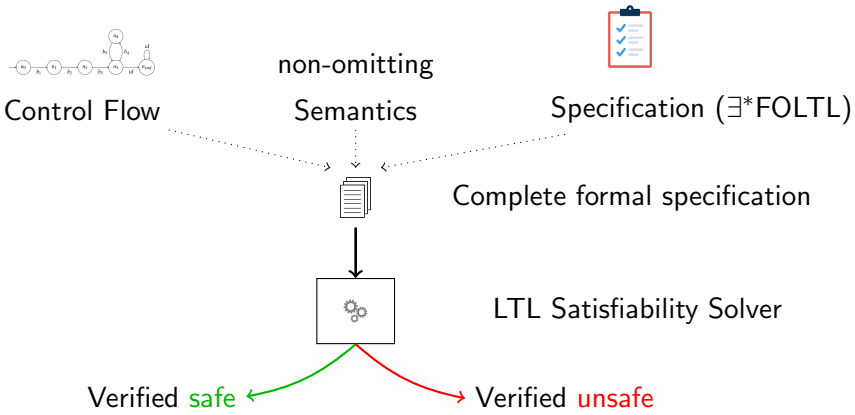
Verification



Verification



Verification



Theorem: Workflow Verification [CCS '17]

$w \models \neg\varphi$ is decidable for non-omitting workflows.

What's the problem?

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

is expressed by

$$\forall x\colon PC, y\colon PC, p\colon Paper, r\colon Review.\ \mathbf{X}\, Read(x, y, p, r) \leftrightarrow$$
$$Read(x, y, p, r) \vee (Assign(x, p) \wedge Review(y, p, r))$$

What's the problem?

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

is expressed by

$$\forall x\colon PC, \qquad p\colon Paper, r\colon Review.\ \mathbf{X}\, Read(x, \quad p, r) \leftrightarrow$$
$$Read(x, \quad p, r) \vee (Assign(x, p) \wedge \exists y\colon PC.Review(y, p, r))$$

What's the problem?

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

is expressed by

$$\forall x\colon PC, \qquad p\colon Paper, r\colon Review.\ \mathbf{X}\, Read(x, \quad p, r) \leftrightarrow$$
$$Read(x, \quad p, r) \vee (Assign(x,p) \wedge \exists y\colon PC.Review(y,p,r))$$

Theorem: Verification for Omitting Workflows [CCS '17]

$w \models \neg\varphi$ is undecidable for omitting workflows even if $\neg\varphi$ is decidable.

Section 3

Information Flow Security

Noninterference

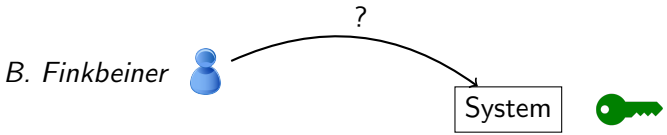
Can any agent learn secret information?

B. Finkbeiner 

System 

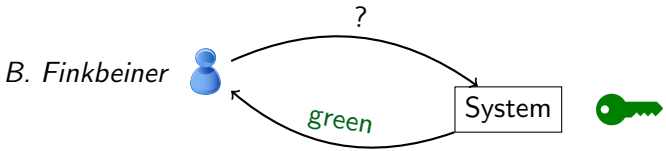
Noninterference

Can any agent learn secret information?



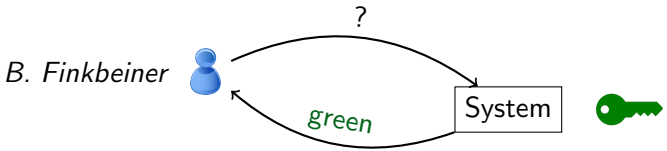
Noninterference

Can any agent learn secret information?



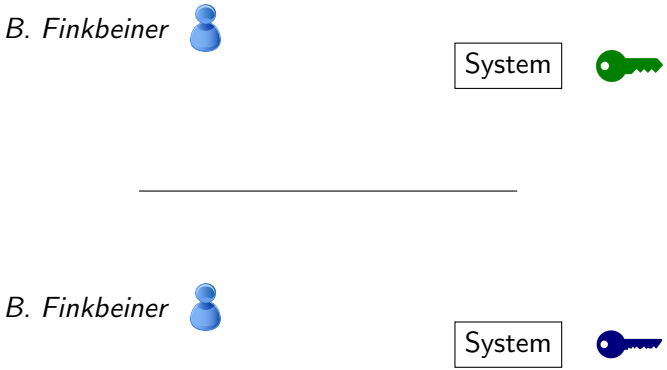
Noninterference

Can any agent learn secret information?

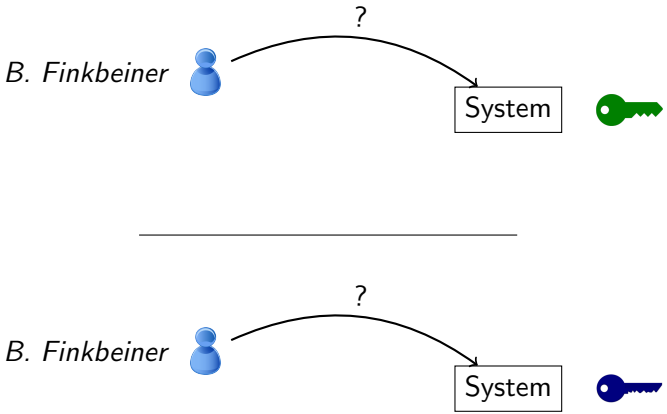


Does *B. Finkbeiner* learn something?

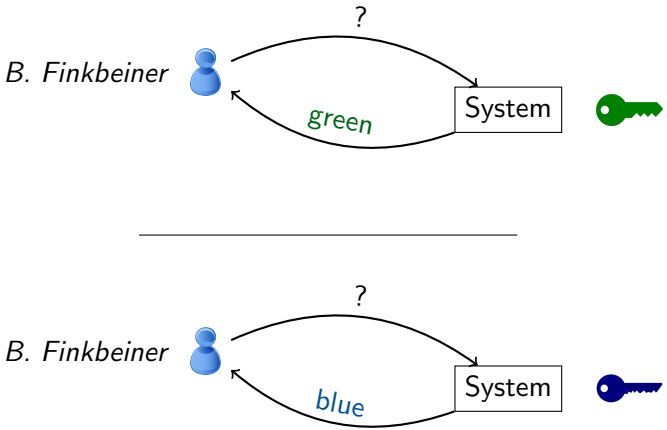
Noninterference — Hyperproperty



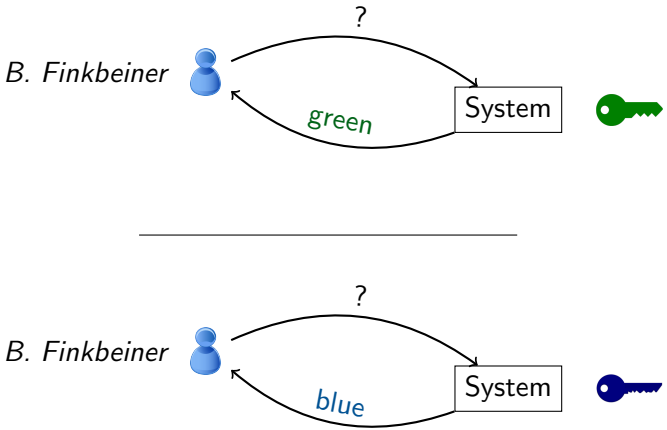
Noninterference — Hyperproperty



Noninterference — Hyperproperty

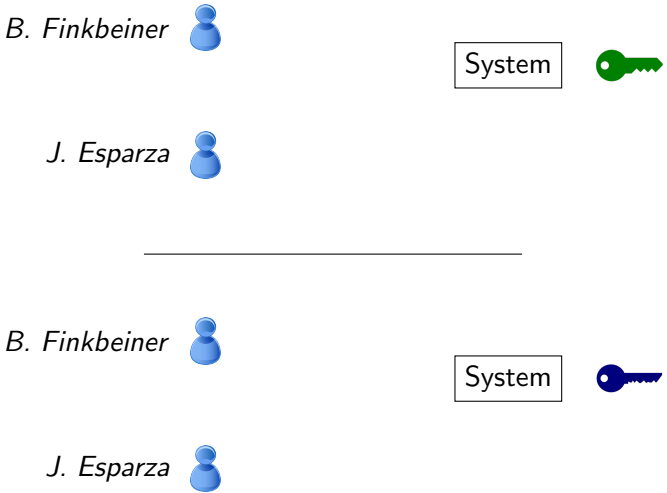


Noninterference — Hyperproperty

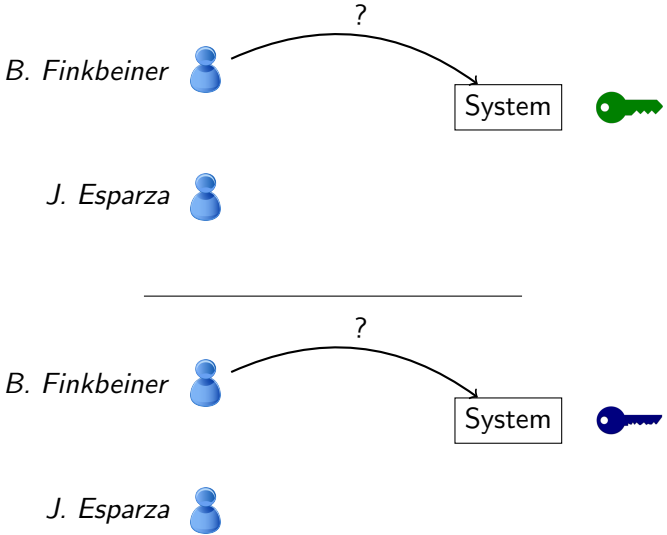


Is this enough?

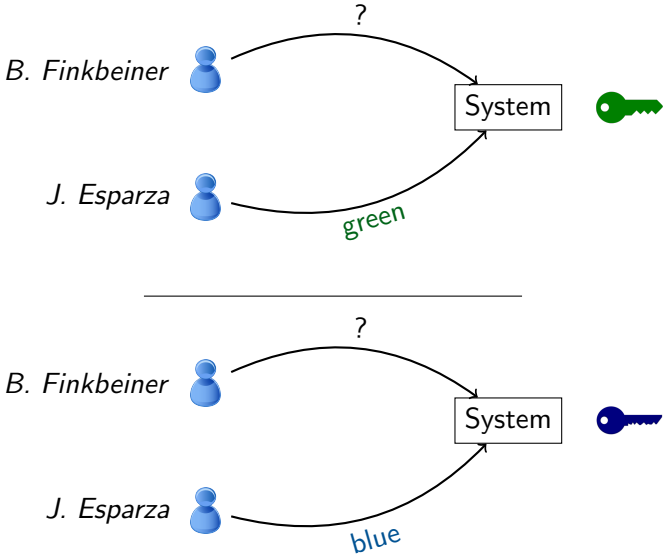
Noninterference



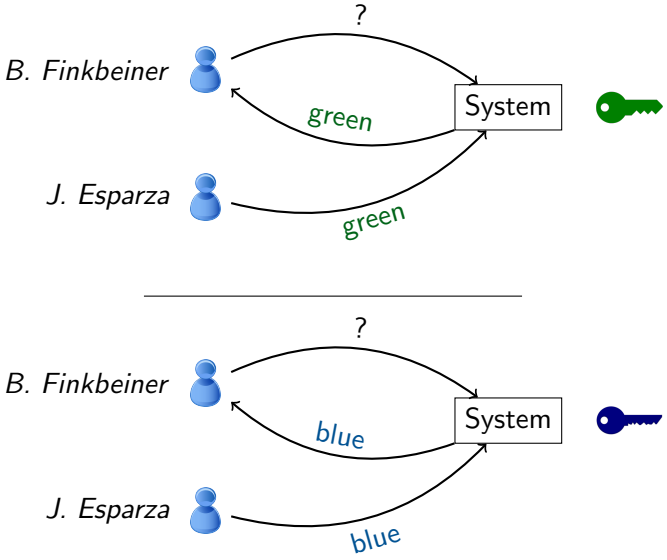
Noninterference



Noninterference



Noninterference



Agent Models

Stubborn Agents

Agent behavior does not depend on secret information.
$\Rightarrow$ Observing other agents does not leak information.

Agent Models

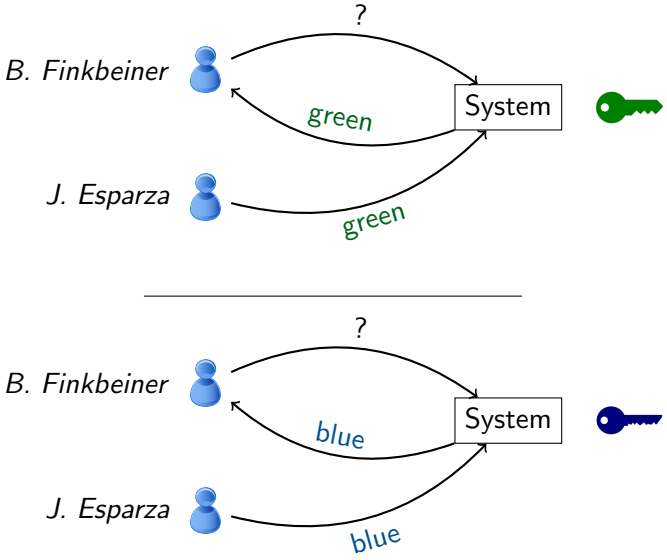
Stubborn Agents

Agent behavior does not depend on secret information.
$\Rightarrow$ Observing other agents does not leak information.

Causal Agents

Agents will actively change their behavior to leak as much information as possible.
$\Rightarrow$ Causal agents conspire against the system.

Noninterference



Verification for Hyperproperties

For any *non-omitting* workflow w and specification in $\exists^*_\pi \forall^*_\pi \exists^*$FOLTL

Theorem: Workflow Verification for Hyperproperties

1. $w \models \neg\varphi$ is decidable for a fixed upper bound of causal agents.
2. $w \models \neg\varphi$ is undecidable for an unlimited amount of causal agents.

Verification for Hyperproperties

For any *non-omitting* workflow w and specification in $\exists^*_\pi \forall^*_\pi \exists^*$FOLTL

Theorem: Workflow Verification for Hyperproperties

1. $w \models \neg\varphi$ is decidable for a fixed upper bound of causal agents.
2. $w \models \neg\varphi$ is undecidable for an unlimited amount of causal agents.

Corollary: Noninterference

Noninterference for a fixed upper bound of causal agents is decidable.

Section 4

Implementation

Nice theory — but does it work?

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

Input

- Non-omitting Workflow specification
- Declassification condition
- Amount and sort of causal agents

Builds the correct first-order formulas, translates to LTL
$\Rightarrow$ Solve using any LTL satisfiability solver.

NIWO — Scalability

Resulting LTL formula

- Scales linearly to larger workflows
- Scales exponentially in the number of causal agents.

NIWO — Scalability

Resulting LTL formula

- Scales linearly to larger workflows
- Scales exponentially in the number of causal agents.

Solving the resulting LTL satisfiability problem

- Only stubborn agents
  $\Rightarrow$ Verifies example workflows in seconds
- Small number of causal agents
  $\Rightarrow$ Verifies example workflows in minutes

Example Revisited

% 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

Example Revisited

% 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)

$\}$

Example Revisited

% PC members may declare conflicts
forall x, p **may**. $true \rightarrow Conflict \mathrel{+}= (x, p)$

Relation	π	π'
Conflict	$(Alice, p)$	

Example Revisited

% PC members may declare conflicts
forall x, p **may**. $true \rightarrow Conflict \mathrel{+}= (x, p)$

% PC members are assigned to papers
forall x, p **may**. $\neg Conflict(x, p) \rightarrow Assign \mathrel{+}= (x, p)$

Relation	π	π'
Conflict	$(Alice, p)$	
Assign	$(Alice, q), (Bob, q), (Bob, p)$	

Example Revisited

% PC members may declare conflicts
forall x, p **may**. $true \rightarrow Conflict \mathrel{+}= (x, p)$

% PC members are assigned to papers
forall x, p **may**. $\neg Conflict(x, p) \rightarrow Assign \mathrel{+}= (x, p)$

% PC members write reviews for papers
forall x, p, r. $Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$

Relation	π	π'
Conflict	$(Alice, p)$	
Assign	$(Alice, q), (Bob, q), (Bob, p)$	
Review	(Bob, p, r_p) (Bob, q, r_q)	(Bob, q, r_q)

Example Revisited

% PC members may declare conflicts
forall x, p **may**. $true \rightarrow Conflict \mathrel{+}= (x, p)$

% PC members are assigned to papers
forall x, p **may**. $\neg Conflict(x, p) \rightarrow Assign \mathrel{+}= (x, p)$

% PC members write reviews for papers
forall x, p, r. $Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$

% PC members read all other reviews
forall x, y, p, r. $Assign(x, p) \wedge Review(y, p, r) \rightarrow Read \mathrel{+}= (x, y, p, r)$

Relation	π	π'
Conflict	$(Alice, p)$	
Assign	$(Alice, q), (Bob, q), (Bob, p)$	
Review	(Bob, p, r_p) (Bob, q, r_q)	(Bob, q, r_q)
Read	$(Alice, Bob, q, r_q)$	

Example Revisited

% PC members may declare conflicts
forall x, p **may**. $true \rightarrow Conflict \mathrel{+}= (x, p)$

% PC members are assigned to papers
forall x, p **may**. $\neg Conflict(x, p) \rightarrow Assign \mathrel{+}= (x, p)$

% PC members write reviews for papers
forall x, p, r. $Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$

% PC members read all other reviews
forall x, y, p, r. $Assign(x, p) \wedge Review(y, p, r) \rightarrow Read \mathrel{+}= (x, y, p, r)$

% PC members can rethink their reviews
forall x, p, r **may**. $Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$

Relation	π	π'
Conflict	$(Alice, p)$	
Assign	$(Alice, q), (Bob, q), (Bob, p)$	
Review	(Bob, p, r_p) (Bob, q, r_q)	(Bob, q, r_q)
Read	$(Alice, Bob, q, r_q)$	
Review	(Bob, q, r_p)	

Example Revisited

% PC members may declare conflicts
forall x, p **may**. $true \rightarrow Conflict \mathrel{+}= (x, p)$

% PC members are assigned to papers
forall x, p **may**. $\neg Conflict(x, p) \rightarrow Assign \mathrel{+}= (x, p)$

% PC members write reviews for papers
forall x, p, r. $Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$

% PC members read all other reviews
forall x, y, p, r. $Assign(x, p) \wedge Review(y, p, r) \rightarrow Read \mathrel{+}= (x, y, p, r)$

% PC members can rethink their reviews
forall x, p, r **may**. $Assign(x, p) \wedge Oracle(x, p, r) \rightarrow Review \mathrel{+}= (x, p, r)$

% PC members read all other reviews
forall x, y, p, r. $Assign(x, p) \wedge Review(y, p, r) \rightarrow Read \mathrel{+}= (x, y, p, r)$

Relation	π	π'
Conflict	$(Alice, p)$	
Assign	$(Alice, q), (Bob, q), (Bob, p)$	
Review	(Bob, p, r_p) (Bob, q, r_q)	(Bob, q, r_q)
Read	$(Alice, Bob, q, r_q)$	
Review	(Bob, q, r_p)	
Read	$(Alice, Bob, q, r_p)$	

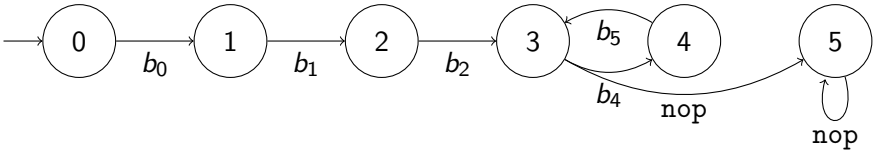
Summary — Non-omitting workflows

1. Developed theory for Workflow Verification
2. Proved (un-)decidability of Workflow Verification for a meaningful fragment of Hyper-Properties (e.g. Noninterference)
3. Built NIWO, an automatic analyzer for Noninterference in non-omitting Workflows

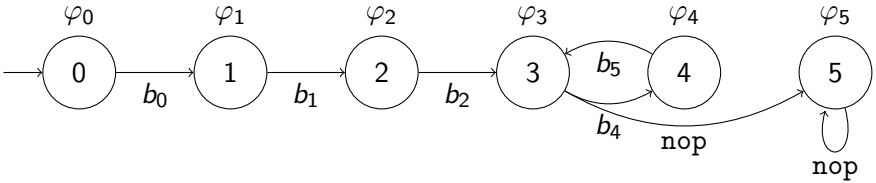
Section 5

Omitting Workflows — Inductive Invariants to the Rescue

Invariants



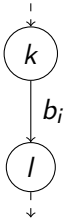
Invariants



Invariant

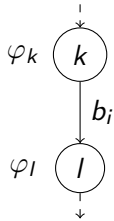
An *invariant* φ is an assignment of locations to first-order formulas φ_i.

Inductive Invariants



b_i : **forall** x: PC, p: $Paper$ **may**. $\neg Conflict(x, p) \rightarrow Assign \mathrel{+}= (x, p)$

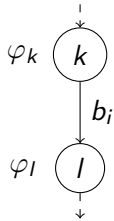
Inductive Invariants



Inductiveness

Whenever the workflow goes from state k to state l and φ_k held before, then φ_l will hold afterwards

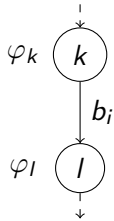
Inductive Invariants



Inductiveness

Whenever the workflow goes from state k to state l and φ_k held before, then φ_l will hold afterwards — i.e. $\varphi_k \rightarrow \mathrm{WP}[\![b_i]\!](\varphi_l)$

Inductive Invariants



Inductiveness

Whenever the workflow goes from state k to state l and φ_k held before, then φ_l will hold afterwards — i.e. $\varphi_k \rightarrow \mathrm{WP}[\![b_i]\!](\varphi_l)$

Theorem: Validity

For an invariant φ where all φ_i are *universal*, it is decidable wether φ holds for a given workflow.

Simplifying the desired properties

Noninterference property consists of

- 2 copies of all relations
- Agent model
- Declassification
- No observable differences

Simplifying the desired properties

Noninterference property consists of

- 2 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

Simplifying the desired properties

Noninterference property consists of

- 2 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

Corollary: Noninterference

Given a suitable universal invariant φ, it is decidable if φ proves that Noninterference holds (again for a fixed upper bound of causal agents)

Summary — Omitting Workflows

Summary

1. It is undecidable if a given FOLTL property holds.
2. Solution: Use inductive universal invariants to show that properties hold.
3. If properties are too complicated, transform the workflow.

What I haven't shown

For a large subclass of omitting workflows:
If there exists a universal invariant that can be used to prove Noninterference,
we can compute it.