CoCon: A Confidentiality-Verified Conference Management System

Andrei Popescu Peter Lammich Thomas Bauereiss

March 17, 2025

Abstract

This entry contains the confidentiality verification of the (functional kernel of) the CoCon conference management system [3, 6]. The confidentiality properties refer to the documents managed by the system, namely papers, reviews, discussion logs and acceptance/rejection decisions, and also to the assignment of reviewers to papers. They have all been formulated as instances of BD Security [4, 5] and verified using the BD Security unwinding technique.

Contents

1 Introduction

This document presents the confidentiality verification of the (functional kernel of) the CoCon conference management system [3, 6]. CoCon was the first case study of BD Security, a general framework for the specification and verification of information flow security. The framework works for any input/output (I/O) automata and allows the specification of flexible policies for information flow security by describing the observations, the secrets, a bound on information release (also known as "declassification bound") and a trigger for information release (also known as "declassification trigger').

In CoCon, a conference goes through several phases:

No-Phase Any user can apply for a new conference, with the effect of registering it in the system as initially having "no phase." After approval from CoCon's superuser,[1] the conference moves to the setup phase, with the applicant becoming a conference chair.

Setup A conference chair can add new chairs and new regular PC members. From here on, advancing the conference through its different phases can be done by the chairs.

Submission Any user can list the conferences awaiting submissions (i.e., being in the submission phase). A user can submit new papers, upload new versions of their existing papers, or indicate other users as coauthors thereby granting them reading and editing rights.

Bidding Authors are no longer allowed to upload or register new papers, and PC members are allowed to view the submitted papers. PC members can place bids, indicating for each paper one of the following preferences: "want to review", "would review", "no preference", "would not review", and "conflict". If the preference is "conflict", the PC

[1] The superuser's powers are restricted to approving or rejecting new conference requests.

member cannot be assigned that paper, and will not see its discussion. “Conflict” is assigned automatically to papers authored by a PC member.

Reviewing Chairs can assign reviewers to papers, which must be among the PC members who have no conflict with given paper. The assigned reviewers can edit their reviews.

Discussion All PC members having no conflict with a paper can see its reviews and can add comments. The reviewers can still edit their reviews, but in a transparent manner—so that the overwritten versions are still visible to the non-conflict PC members. Also, chairs can edit the decision.

Notification The authors can read the reviews and the accept/reject decision, which no one can edit any longer.

After this introduction and a section on technical preliminaries, this document presents the specification of the CoCon system, as an input/output (I/O) automaton. Following is a section on proved safety properties about the system (invariants) that are needed in the proofs of confidentiality.

The confidentiality properties of CoCon are formalized as instances of BD Security. They cover confidentiality aspects about:

- papers
- reviews of papers
- discussion logs consisting of comments from the PC members
- decisions on the papers’ acceptance or rejection
- assignment of reviewers to papers

Each of these types of confidentiality properties have dedicated sections (and corresponding folders in the formalization) with self-explanatory names. BD Security is defined in terms of an observation infrastructure, a secrecy infrastructure, a declassification trigger and a declassification bound. The observations are always given by an arbitrary set of users (which is fixed in the “Observation Setup” section). The secrets (called “values” in this formalization) and the declassification bounds and triggers are specific to each property. The bounds and triggers are chosen in such a way that their interplay covers the entire spectrum of information flow through the system in relation to the given secrets. This is explained in [6, Section 3.5].

The proofs proceed using the method of BD Security unwinding, which is part of the AFP entry on BD Security [5] and is described in detail in [6, Sections 4.1 and 4.2] and [4, Sections 2.5 and 2.6]. For managing proof complexity, we take a modular approach, building several unwinding relations that are connected in a sequence and have an exit point into an error component. This approach is presented in [6] as Corollary 6 (Sequential Unwinding Theorem) and in [4] as Theorem 4 (Sequential Multiplex Unwinding Theorem).

The last section formalizes what we call *traceback properties*,[2] which strengthen the confidentiality guarantees. Confidentiality formulated as BD security states properties of essentially the following form: "Unless a user acquires such and such role and/or the conference reaches such and such phase, that user cannot learn such and such information." Traceback properties show that it is not possible for a user to usurp such roles, and that the conference only progresses through different phases in a "legal" way. [6, Section 3.6] explains CoCon's traceback properties in detail.

As a matter of notation, this formalization (similarly to all our AFP formalizations involving BD security) concurs with the original conference paper on CoCon [3] and differs from the later journal paper [6] in that the secrets are called "values" (and consequently the type of secrets is denoted by "value"), and are ranged over by "v" rather than "s". On the other hand, we use "s" (rather than "σ") to range over states. Moreover, the formalization uses the following notations for the various BD security components:

- phi for the secret discriminator isSec
- f for the secret selector getSec
- gamma for the observation discriminator isObs
- g for the observation selector getObs

theory *Prelim*
  imports *Fresh-Identifiers.Fresh-String Bounded-Deducibility-Security.Trivia*
begin

1.1 The basic types

type-synonym *string* = *String.literal*
definition *emptyStr* = *STR ''''*

type-synonym *phase* = *nat*

abbreviation *noPH* ≡ (*0::nat*) **abbreviation** *setPH* ≡ *Suc noPH* **abbreviation** *subPH* ≡ *Suc setPH*
abbreviation *bidPH* ≡ *Suc subPH* **abbreviation** *revPH* ≡ *Suc bidPH* **abbreviation** *disPH* ≡ *Suc revPH*

[2] In previous work, we called these types of properties *accountability properties* [1, 2] or *forensic properties* [3]. The *traceback properties* terminology is used in [6].

abbreviation *notifPH* ≡ *Suc disPH* **abbreviation** *closedPH* ≡ *Suc notifPH*

fun *SucPH* **where**
SucPH ph = (*if ph* = *closedPH then closedPH else Suc ph*)

datatype *user* = *User string string string*
fun *nameUser* **where** *nameUser* (*User name info email*) = *name*
fun *infoUser* **where** *infoUser* (*User name info email*) = *info*
fun *emailUser* **where** *emailUser* (*User name info email*) = *email*
definition *emptyUser* ≡ *User emptyStr emptyStr emptyStr*

typedecl *raw-data*
code-printing type-constructor *raw-data* ⇀ (*Scala*) *java.io.File*

datatype *pcontent* = *NoPContent* | *PContent raw-data*

datatype *score* = *NoScore* | *MinusThree* | *MinusTwo* | *MinusOne* | *Zero* | *One* | *Two* | *Three*

fun *scoreAsInt* :: *score* ⇒ *int* **where**
scoreAsInt MinusThree = −*3*
|*scoreAsInt MinusTwo* = −*2*
|*scoreAsInt MinusOne* = −*1*
|*scoreAsInt Zero* = *0*
|*scoreAsInt One* = *1*
|*scoreAsInt Two* = *2*
|*scoreAsInt Three* = *3*

datatype *exp* = *NoExp* | *Zero* | *One* | *Two* | *Three* | *Four*

type-synonym *rcontent* = *exp* ∗ *score* ∗ *string*
fun *scoreOf* :: *rcontent* ⇒ *score* **where** *scoreOf* (*exp*,*sc*,*txt*) = *sc*

type-synonym *review* = *rcontent list*

abbreviation *emptyReview* :: *review* **where** *emptyReview* ≡ []
datatype *discussion* = *Dis string list*
definition *emptyDis* ≡ *Dis* []
datatype *decision* = *NoDecision* | *Accept* | *Reject*

datatype *paper* = *Paper string string pcontent review list discussion decision list*

fun *titlePaper* **where** *titlePaper* (*Paper title abstract content reviews dis decs*) = *title*

fun *abstractPaper* **where** *abstractPaper* (*Paper title abstract content reviews dis decs*) = *abstract*
fun *contentPaper* **where** *contentPaper* (*Paper title abstract content reviews dis decs*) = *content*
fun *reviewsPaper* **where** *reviewsPaper* (*Paper title abstract content reviews dis decs*) = *reviews*
fun *disPaper* **where** *disPaper* (*Paper title abstract content reviews dis decs*) = *dis*

fun *decsPaper* **where** *decsPaper* (*Paper title abstract content reviews dis decs*) = *decs*

fun *decPaper* **where** *decPaper pap* = *hd* (*decsPaper pap*)

definition *emptyPaper* :: *paper* **where**
emptyPaper ≡ *Paper emptyStr emptyStr NoPContent* [] *emptyDis* []

datatype *conf* = *Conf string string*
fun *nameConf* **where** *nameConf* (*Conf name info*) = *name*
fun *infoConf* **where** *infoConf* (*Conf name info*) = *info*
definition *emptyConf* ≡ *Conf emptyStr emptyStr*

datatype *password* = *Password string*
definition *emptyPass* ≡ *Password emptyStr*

datatype *preference* = *NoPref* | *Want* | *Would* | *WouldNot* | *Conflict*

1.2 Conference, user and paper IDs

datatype *userID* = *UserID string*
datatype *paperID* = *PaperID string*
datatype *confID* = *ConfID string*

definition *emptyUserID* ≡ *UserID emptyStr*
definition *voronkovUserID* ≡ *UserID* (*STR ''voronkov''*)
definition *emptyPaperID* ≡ *PaperID emptyStr*
definition *emptyConfID* ≡ *ConfID emptyStr*

datatype *role* = *Aut paperID* | *Rev paperID nat* | *PC* | *Chair*
fun *isRevRole* **where** *isRevRole* (*Rev* - -) = *True* | *isRevRole* - = *False*

fun *isRevRoleFor* :: *paperID* ⇒ *role* ⇒ *bool* **where**
isRevRoleFor papID (*Rev papID' n*) ⟷ *papID* = *papID'*
|*isRevRoleFor papID* - ⟷ *False*

fun *userIDAsStr* **where** *userIDAsStr* (*UserID str*) = *str*

definition *getFreshUserID userIDs* ≡ *UserID* (*fresh* (*set* (*map userIDAsStr userIDs*)) (*STR ''1''*))

lemma *UserID-userIDAsStr*[*simp*]: *UserID* (*userIDAsStr userID*) = *userID*
⟨*proof*⟩

lemma *member-userIDAsStr-iff* [*simp*]: *str* ∈ *userIDAsStr* ‘ (*set userIDs*) ⟷ *UserID str* ∈∈ *userIDs*
⟨*proof*⟩

lemma *getFreshUserID*: ¬ *getFreshUserID userIDs* ∈∈ *userIDs*
⟨*proof*⟩

instantiation *userID* :: *linorder*
begin
definition *le-userID-def*: *uid* ≤ *uid'* ≡ *case* (*uid*,*uid'*) *of* (*UserID str*, *UserID str'*) ⇒ *str* ≤ *str'*
definition *lt-userID-def*: *uid* < *uid'* ≡ *case* (*uid*,*uid'*) *of* (*UserID str*, *UserID str'*) ⇒ *str* < *str'*
instance ⟨*proof*⟩
end

fun *paperIDAsStr* **where** *paperIDAsStr* (*PaperID str*) = *str*

definition *getFreshPaperID paperIDs* ≡ *PaperID* (*fresh* (*set* (*map paperIDAsStr paperIDs*)) (*STR ''2''*))

lemma *PaperID-paperIDAsStr*[*simp*]: *PaperID* (*paperIDAsStr paperID*) = *paperID*
⟨*proof*⟩

lemma *member-paperIDAsStr-iff* [*simp*]: *str* ∈ *paperIDAsStr* ‘ *paperIDs* ⟷ *PaperID str* ∈ *paperIDs*
⟨*proof*⟩

lemma *getFreshPaperID*: ¬ *getFreshPaperID paperIDs* ∈∈ *paperIDs*
⟨*proof*⟩

instantiation *paperID* :: *linorder*
begin
definition *le-paperID-def*: *uid* ≤ *uid'* ≡ *case* (*uid*,*uid'*) *of* (*PaperID str*, *PaperID str'*) ⇒ *str* ≤ *str'*
definition *lt-paperID-def*: *uid* < *uid'* ≡ *case* (*uid*,*uid'*) *of* (*PaperID str*, *PaperID str'*) ⇒ *str* < *str'*
instance ⟨*proof*⟩
end

fun *confIDAsStr* **where** *confIDAsStr* (*ConfID str*) = *str*

definition *getFreshConfID confIDs* ≡ *ConfID* (*fresh* (*set* (*map confIDAsStr confIDs*)) (*STR ''2''*))

lemma *ConfID-confIDAsStr*[*simp*]: *ConfID* (*confIDAsStr confID*) = *confID*
⟨*proof*⟩

lemma *member-confIDAsStr-iff*[*simp*]: *str* ∈ *confIDAsStr* ‘ (*set confIDs*) ⟷ *ConfID str* ∈∈ *confIDs*
⟨*proof*⟩

lemma *getFreshConfID*: ¬ *getFreshConfID confIDs* ∈∈ *confIDs*
⟨*proof*⟩

instantiation *confID* :: *linorder*
begin
definition *le-confID-def*: *uid* ≤ *uid'* ≡ *case* (*uid*,*uid'*) *of* (*ConfID str*, *ConfID str'*) ⇒ *str* ≤ *str'*
definition *lt-confID-def*: *uid* < *uid'* ≡ *case* (*uid*,*uid'*) *of* (*ConfID str*, *ConfID str'*) ⇒ *str* < *str'*
instance ⟨*proof*⟩
end

end

2 System specification

This section formalizes the CoCon system as an I/O automaton. We call the inputs "actions".

theory *System-Specification*
imports *Prelim*
begin

2.1 System state

The superuser of the system is called "voronkov", as a form acknowledgement for our inspiration from EasyChair when creating CoCon.

record *state* =
confIDs :: *confID list*
conf :: *confID* ⇒ *conf*

userIDs :: *userID list*
pass :: *userID* ⇒ *password*
user :: *userID* ⇒ *user*

roles :: *confID* ⇒ *userID* ⇒ *role list*

paperIDs :: *confID* ⇒ *paperID list*
paper :: *paperID* ⇒ *paper*

pref :: *userID* ⇒ *paperID* ⇒ *preference*

voronkov :: *userID*

news :: *confID* ⇒ *string list*
phase :: *confID* ⇒ *phase*

abbreviation *isPC* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**
isPC s confID uID ≡ *PC* ∈∈ *roles s confID uID*
abbreviation *isChair* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**
isChair s confID uID ≡ *Chair* ∈∈ *roles s confID uID*
abbreviation *isAut* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *bool* **where**
isAut s confID uID papID ≡ *Aut papID* ∈∈ *roles s confID uID*
definition *isAutSome* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *bool* **where**
isAutSome s confID uID ≡ *list-ex* (*isAut s confID uID*) (*paperIDs s confID*)

definition *authors* :: *state* ⇒ *confID* ⇒ *paperID* ⇒ *userID list* **where**
authors s confID papID ≡ *filter* (λ *uID*. *isAut s confID uID papID*) (*userIDs s*)
abbreviation *isRevNth* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *nat* ⇒ *bool*
where
isRevNth s confID uID papID n ≡ *Rev papID n* ∈∈ *roles s confID uID*
definition *isRev* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *bool* **where**
isRev s confID uID papID ≡ *list-ex* (*isRevRoleFor papID*) (*roles s confID uID*)

definition *getRevRole* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *role option*
where
getRevRole s confID uID papID ≡ *List.find* (*isRevRoleFor papID*) (*roles s confID uID*)

definition *getNthReview* :: *state* ⇒ *paperID* ⇒ *nat* ⇒ *review* **where**
getNthReview s papID n ≡ (*reviewsPaper* (*paper s papID*))!*n*

definition *getAllPaperIDs* :: *state* ⇒ *paperID list* **where**
getAllPaperIDs s ≡ *concat* [*paperIDs s confID*. *confID* ← *confIDs s*]
definition *getReviewIndex* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *paperID* ⇒ *nat* **where**
getReviewIndex s confID uID papID ≡
case getRevRole s confID uID papID of Some (*Rev papID′ n*) ⇒ *n*

definition *getReviewersReviews* :: *state* ⇒ *confID* ⇒ *paperID* ⇒ (*userID* ∗ *review*) *list* **where**
getReviewersReviews s confID papID ≡
[(*uID*, *getNthReview s papID* (*getReviewIndex s confID uID papID*)).
uID ← *userIDs s*,
isRev s confID uID papID
]

definition *isAUT* :: *state* ⇒ *userID* ⇒ *paperID* ⇒ *bool* **where**
isAUT s uID papID ≡ ∃ *confID*. *isAut s confID uID papID*
definition *isREVNth* :: *state* ⇒ *userID* ⇒ *paperID* ⇒ *nat* ⇒ *bool* **where**
isREVNth s uID papID n ≡ ∃ *confID*. *isRevNth s confID uID papID n*

lemma *isRev-getRevRole*:
assumes *isRev s confID uID papID*
shows *getRevRole s confID uID papID* ≠ *None*
⟨*proof*⟩

lemma *getRevRole-Some*:
assumes *getRevRole s confID uID papID* = *Some role*
shows ∃ *n*. *role* = *Rev papID n*
⟨*proof*⟩

lemma *isRev-getRevRole2*:
assumes *isRev s confID uID papID***shows** ∃ *n*. *getRevRole s confID uID papID* = *Some* (*Rev papID n*)
⟨*proof*⟩

lemma *isRev-imp-isRevNth-getReviewIndex*:
assumes *isRev s confID uID papID*
shows *isRevNth s confID uID papID* (*getReviewIndex s confID uID papID*)
⟨*proof*⟩

lemma *isRev-def2*:
isRev s confID uID papID ⟷ (∃ *n*. *isRevNth s confID uID papID n*) (**is** *?A* ⟷ *?B*)
⟨*proof*⟩

lemma *isRev-def3*:
isRev s confID uID papID ⟷ *isRevNth s confID uID papID* (*getReviewIndex s confID uID papID*)
⟨*proof*⟩

lemma *getFreshPaperID-getAllPaperIDs*[*simp*]:
assumes *confID* ∈∈ *confIDs s*

shows $\neg$ *getFreshPaperID* (*getAllPaperIDs s*) $\in\in$ *paperIDs s confID*
⟨*proof*⟩

lemma *getRevRole-Some-Rev*:
getRevRole s cid uid pid = *Some* (*Rev pid′ n*) $\Longrightarrow$ *pid′* = *pid*
⟨*proof*⟩

lemma *getRevRole-Some-Rev-isRevNth*:
getRevRole s cid uid pid = *Some* (*Rev pid′ n*) $\Longrightarrow$ *isRevNth s cid uid pid n*
⟨*proof*⟩

definition *IDsOK* :: *state* ⇒ *confID list* ⇒ *userID list* ⇒ *paperID list* ⇒ *bool*
where
IDsOK s cIDs uIDs papIDs ≡
list-all (λ *confID*. *confID* ∈∈ *confIDs s*) *cIDs* ∧
list-all (λ *uID*. *uID* ∈∈ *userIDs s*) *uIDs* ∧
list-all (λ *papID*. *papID* ∈∈ *paperIDs s* (*hd cIDs*)) *papIDs*

2.2 The actions

2.2.1 Initialization of the system

definition *istate* :: *state*
where
istate ≡
⦇
confIDs = [],
conf = (λ *confID*. *emptyConf*),
userIDs = [*voronkovUserID*],
pass = (λ *uID*. *emptyPass*),
user = (λ *uID*. *emptyUser*),
roles = (λ *confID uID*. []),
paperIDs = (λ *confID*. []),
paper = (λ *papID*. *emptyPaper*),
pref = (λ *uID papID*. *NoPref*),
voronkov = *voronkovUserID*,
news = (λ *confID*. []),
phase = (λ *confID*. *noPH*)
⦈

2.2.2 Actions unbound by any existing conference (with its phases)

definition *createUser* :: *state* ⇒ *userID* ⇒ *password* ⇒ *string* ⇒ *string* ⇒ *string* ⇒ *state*
where
createUser s uID p name info email ≡
let uIDs = *userIDs s*
in
s ⦇*userIDs* := *uID* # *uIDs*,

$user := (user\ s)\ (uID := User\ name\ info\ email),$
$pass := (pass\ s)\ (uID := p)\rparr$

definition $e\text{-}createUser :: state \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow bool$ **where**
$e\text{-}createUser\ s\ uID\ p\ name\ info\ email \equiv$
$\neg\ uID \in\in userIDs\ s$

definition $updateUser :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow state$
where
$updateUser\ s\ uID\ p\ p'\ name\ info\ email \equiv$
$s\ \lparr user := (user\ s)\ (uID := User\ name\ info\ email),$
$pass := (pass\ s)\ (uID := p')\rparr$

definition $e\text{-}updateUser :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow string \Rightarrow bool$
where
$e\text{-}updateUser\ s\ uID\ p\ p'\ name\ info\ email \equiv$
$IDsOK\ s\ []\ [uID]\ []\ \wedge\ pass\ s\ uID = p$

definition $readAmIVoronkov :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$readAmIVoronkov\ s\ uID\ p \equiv$
$uID = voronkov\ s$

definition $e\text{-}readAmIVoronkov :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}readAmIVoronkov\ s\ uID\ p \equiv$
$IDsOK\ s\ []\ [uID]\ []\ \wedge\ pass\ s\ uID = p$

definition $readUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow string * string * string$
where
$readUser\ s\ uID\ p\ uID' \equiv$
$case\ user\ s\ uID'\ of\ User\ name\ info\ email \Rightarrow (name,\ info,\ email)$

definition $e\text{-}readUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
$e\text{-}readUser\ s\ uID\ p\ uID' \equiv$
$IDsOK\ s\ []\ [uID,uID']\ []\ \wedge\ pass\ s\ uID = p$

definition $createConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow state$
where
$createConf\ s\ confID\ uID\ p\ name\ info \equiv$
$let\ confIDs = confIDs\ s$
in
$s\ (|confIDs := confID\ \#\ confIDs,$
$conf := (conf\ s)\ (confID := Conf\ name\ info),$
$roles := fun\text{-}upd2\ (roles\ s)\ confID\ uID\ [PC,Chair]$
$|)$

definition $e\text{-}createConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow string \Rightarrow bool$
where
$e\text{-}createConf\ s\ confID\ uID\ p\ name\ info \equiv$
$IDsOK\ s\ []\ [uID]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$\neg\ confID \in\in confIDs\ s$

definition $readConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string * string * (role\ list) * phase$
where
$readConf\ s\ confID\ uID\ p \equiv$
$(nameConf\ (conf\ s\ confID),\ infoConf\ (conf\ s\ confID),$
$[rl \leftarrow roles\ s\ confID\ uID.\ \neg\ isRevRole\ rl],\ phase\ s\ confID)$

definition $e\text{-}readConf :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}readConf\ s\ confID\ uID\ p \equiv$
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge\ pass\ s\ uID = p$

definition $listConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$
where
$listConfs\ s\ uID\ p \equiv$
$confIDs\ s$

definition $e\text{-}listConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}listConfs\ s\ uID\ p \equiv$
$IDsOK\ s\ []\ [uID]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$uID = voronkov\ s$

definition $listAConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$
where

$listAConfs\ s\ uID\ p \equiv$
$[confID.\ confID \leftarrow confIDs\ s,\ phase\ s\ confID = noPH]$

definition $e\text{-}listAConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}listAConfs\ s\ uID\ p \equiv$
$IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p \wedge$
$uID = voronkov\ s$

definition $listSConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$
where
$listSConfs\ s\ uID\ p \equiv$
$[confID.\ confID \leftarrow confIDs\ s,\ phase\ s\ confID = subPH]$

definition $e\text{-}listSConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}listSConfs\ s\ uID\ p \equiv$
$IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p$

definition $listMyConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow confID\ list$
where
$listMyConfs\ s\ uID\ p \equiv$
$[confID.\ confID \leftarrow confIDs\ s\ ,\ roles\ s\ confID\ uID \neq []]$

definition $e\text{-}listMyConfs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}listMyConfs\ s\ uID\ p \equiv$
$IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p$

definition $listAllUsers :: state \Rightarrow userID \Rightarrow password \Rightarrow userID\ list$
where
$listAllUsers\ s\ uID\ p \equiv$
$userIDs\ s$

definition $e\text{-}listAllUsers :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}listAllUsers\ s\ uID\ p \equiv$
$IDsOK\ s\ []\ [uID]\ [] \wedge pass\ s\ uID = p$

definition $listAllPapers :: state \Rightarrow userID \Rightarrow password \Rightarrow paperID\ list$
where
$listAllPapers\ s\ uID\ p \equiv$
$getAllPaperIDs\ s$

definition $e\text{-}listAllPapers :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

where
$e\text{-}listAllPapers\ s\ uID\ p \equiv$
$IDsOK\ s\ []\ [uID]\ []\ \wedge\ pass\ s\ uID = p$

2.2.3 Actions available in the noPH phase

definition $updateConfA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow state$
where
$updateConfA\ s\ confID\ uID\ p \equiv$
$s\ (\!|phase := (phase\ s)\ (confID := setPH)|\!)$

definition $e\text{-}updateConfA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}updateConfA\ s\ confID\ uID\ p \equiv$
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$uID = voronkov\ s\ \wedge\ phase\ s\ confID = noPH$

2.2.4 Actions available in the setPH phase

definition $createPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow state$
where
$createPC\ s\ confID\ uID\ p\ uID' \equiv$
$let\ rls = roles\ s\ confID\ uID'$
in
$s\ (\!|roles := fun\text{-}upd2\ (roles\ s)\ confID\ uID'\ (List.insert\ PC\ rls)|\!)$

definition $e\text{-}createPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
$e\text{-}createPC\ s\ confID\ uID\ p\ uID' \equiv$
$let\ uIDs = userIDs\ s$
in
$IDsOK\ s\ [confID]\ [uID,uID']\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$phase\ s\ confID = setPH\ \wedge\ isChair\ s\ confID\ uID$

definition $createChair :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow state$
where
$createChair\ s\ confID\ uID\ p\ uID' \equiv$
$let\ rls = roles\ s\ confID\ uID'$
in
$s\ (\!|roles := fun\text{-}upd2\ (roles\ s)\ confID\ uID'\ (List.insert\ PC\ (List.insert\ Chair\ rls))|\!)$

definition $e\text{-}createChair :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
$e\text{-}createChair\ s\ confID\ uID\ p\ uID' \equiv$
$let\ uIDs = userIDs\ s$
in
$IDsOK\ s\ [confID]\ [uID,uID']\ []\ \wedge\ pass\ s\ uID = p\ \wedge$

$phase\ s\ confID = setPH \wedge isChair\ s\ confID\ uID$

2.2.5 Actions available starting from the setPH phase

definition $updatePhase :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow phase \Rightarrow state$
where
$updatePhase\ s\ confID\ uID\ p\ ph \equiv$
$s\ (\!|phase := (phase\ s)\ (confID := ph)|\!)$

definition $e\text{-}updatePhase :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow phase \Rightarrow bool$ **where**
$e\text{-}updatePhase\ s\ confID\ uID\ p\ ph \equiv$
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge pass\ s\ uID = p\ \wedge$
$phase\ s\ confID \geq setPH \wedge phase\ s\ confID < closedPH \wedge isChair\ s\ confID\ uID\ \wedge$
$ph = SucPH\ (phase\ s\ confID)$

definition $uupdateNews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow state$
where
$uupdateNews\ s\ confID\ uID\ p\ ev \equiv$
$let\ evs = news\ s\ confID$
in
$s\ (\!|news := (news\ s)\ (confID := ev\ \#\ evs)|\!)$

definition $e\text{-}uupdateNews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string \Rightarrow bool$
where
$e\text{-}uupdateNews\ s\ confID\ uID\ p\ ev \equiv$
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge pass\ s\ uID = p\ \wedge$
$phase\ s\ confID \geq setPH \wedge phase\ s\ confID < closedPH \wedge isChair\ s\ confID\ uID$

definition $readNews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow string\ list$
where
$readNews\ s\ confID\ uID\ p \equiv$
$news\ s\ confID$

definition $e\text{-}readNews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}readNews\ s\ confID\ uID\ p \equiv$
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge pass\ s\ uID = p\ \wedge$
$phase\ s\ confID \geq setPH \wedge isPC\ s\ confID\ uID$

definition $listPC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow userID\ list$
where
$listPC\ s\ confID\ uID\ p \equiv$

$[uID.\ uID \leftarrow userIDs\ s,\ isPC\ s\ confID\ uID]$

definition *e-listPC* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listPC s confID uID p ≡
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$(phase\ s\ confID \geq subPH \vee (phase\ s\ confID \geq setPH \wedge isChair\ s\ confID\ uID))$

definition *listChair* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *userID list*
where
listChair s confID uID p ≡
$[uID.\ uID \leftarrow userIDs\ s,\ isChair\ s\ confID\ uID]$

definition *e-listChair* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listChair s confID uID p ≡
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$(phase\ s\ confID \geq subPH \vee (phase\ s\ confID \geq setPH \wedge isChair\ s\ confID\ uID))$

2.2.6 Actions available in the subPH phase

definition *createPaper* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *string* ⇒ *string* ⇒ *state*
where
createPaper s confID uID p papID title abstract ≡
let papIDs = paperIDs s confID;
rls = roles s confID uID
in
$s\ (\!|paperIDs := (paperIDs\ s)\ (confID := papID\ \#\ papIDs),$
$paper := (paper\ s)\ (papID := Paper\ title\ abstract\ NoPContent\ []\ (Dis\ [])\ []),$
$roles := fun\text{-}upd2\ (roles\ s)\ confID\ uID\ (List.insert\ (Aut\ papID)\ rls),$
$pref := fun\text{-}upd2\ (pref\ s)\ uID\ papID\ Conflict|\!)$

definition *e-createPaper* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *string* ⇒ *string* ⇒ *bool*
where
e-createPaper s confID uID p papID name info ≡
$IDsOK\ s\ [confID]\ [uID]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$phase\ s\ confID = subPH\ \wedge$
$\neg\ papID \in\in getAllPaperIDs\ s$

definition *createAuthor* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *userID* ⇒ *state*
where
createAuthor s confID uID p papID uID' ≡
let rls = roles s confID uID'

in
$s\ (\!|roles := fun\text{-}upd2\ (roles\ s)\ confID\ uID'\ (List.insert\ (Aut\ papID)\ rls),$
$\quad pref := fun\text{-}upd2\ (pref\ s)\ uID'\ papID\ Conflict|\!)$

definition $e\text{-}createAuthor :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow bool$
where
$e\text{-}createAuthor\ s\ confID\ uID\ p\ papID\ uID' \equiv$
$IDsOK\ s\ [confID]\ [uID,uID']\ [papID] \wedge pass\ s\ uID = p \wedge$
$phase\ s\ confID = subPH \wedge isAut\ s\ confID\ uID\ papID \wedge uID \neq uID'$

definition $updatePaperTA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow string \Rightarrow string \Rightarrow state$
where
$updatePaperTA\ s\ confID\ uID\ p\ papID\ title\ abstract \equiv$
$case\ paper\ s\ papID\ of\ Paper\ title'\ abstract'\ pc\ reviews\ dis\ decs \Rightarrow$
$s\ (\!|paper := (paper\ s)\ (papID := Paper\ title\ abstract\ pc\ reviews\ dis\ decs)|\!)$

definition $e\text{-}updatePaperTA :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow string \Rightarrow string \Rightarrow bool$
where
$e\text{-}updatePaperTA\ s\ confID\ uID\ p\ papID\ name\ info \equiv$
$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$
$phase\ s\ confID = subPH \wedge isAut\ s\ confID\ uID\ papID$

definition $updatePaperC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow pcontent \Rightarrow state$
where
$updatePaperC\ s\ confID\ uID\ p\ papID\ pc \equiv$
$case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc'\ reviews\ dis\ decs \Rightarrow$
$s\ (\!|paper := (paper\ s)\ (papID := Paper\ title\ abstract\ pc\ reviews\ dis\ decs)|\!)$

definition $e\text{-}updatePaperC :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow pcontent \Rightarrow bool$
where
$e\text{-}updatePaperC\ s\ confID\ uID\ p\ papID\ pc \equiv$
$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$
$phase\ s\ confID = subPH \wedge isAut\ s\ confID\ uID\ papID$

definition $createConflict :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow userID \Rightarrow state$
where
$createConflict\ s\ confID\ uID\ p\ papID\ uID' \equiv$
$s\ (\!|pref := fun\text{-}upd2\ (pref\ s)\ uID'\ papID\ Conflict|\!)$

definition *e-createConflict* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *userID* ⇒ *bool*
where
e-createConflict s confID uID p papID uID′ ≡
IDsOK s [*confID*] [*uID*,*uID′*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID = *subPH* ∧ *isAut s confID uID papID* ∧ *isPC s confID uID′*

2.2.7 Actions available starting from the subPH phase

definition *readPaperTAA* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ (*string* ∗ *string* ∗ *userID list*)
where
readPaperTAA s confID uID p papID ≡
case paper s papID of Paper title abstract pc reviews dis decs ⇒
(*title*, *abstract*, [*uID*. *uID* ← *userIDs s* , *isAut s confID uID papID*])

definition *e-readPaperTAA* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*
where
e-readPaperTAA s confID uID p papID ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *subPH* ∧ (*isAut s confID uID papID* ∨ *isPC s confID uID*)

definition *readPaperC* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *pcontent*
where
readPaperC s confID uID p papID ≡
case paper s papID of Paper title abstract pc reviews dis decs ⇒ *pc*

definition *e-readPaperC* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*
where
e-readPaperC s confID uID p papID ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
(
phase s confID ≥ *subPH* ∧ *isAut s confID uID papID* ∨
phase s confID ≥ *bidPH* ∧ *isPC s confID uID*
)

definition *listPapers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID list*
where
listPapers s confID uID p ≡
let paps = *paper s in*
[*papID*. *papID* ← *paperIDs s confID*]

definition *e-listPapers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listPapers s confID uID p ≡
IDsOK s [*confID*] [*uID*] [] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *subPH* ∧ *isPC s confID uID*

definition *listMyPapers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID list*
where
listMyPapers s confID uID p ≡
let paps = *paper s in*
[*papID*. *papID* ← *paperIDs s confID*, *isAut s confID uID papID*]

definition *e-listMyPapers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listMyPapers s confID uID p ≡
IDsOK s [*confID*] [*uID*] [] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *subPH*

2.2.8 Actions available in the bidPH phase

definition *updatePref* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *preference* ⇒ *state*
where
updatePref s confID uID p papID pr ≡
s (|*pref* := *fun-upd2* (*pref s*) *uID papID pr*|)

definition *e-updatePref* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *preference* ⇒ *bool*
where
e-updatePref s confID uID p papID pr ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID = *bidPH* ∧ *isPC s confID uID* ∧
¬ *isAut s confID uID papID*

2.2.9 Actions available starting from the bidPH phase

definition *readPref* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *preference*
where
readPref s confID uID p papID ≡
pref s uID papID

definition *e-readPref* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*
where
e-readPref s confID uID p papID ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *bidPH* ∧ *isPC s confID uID*

2.2.10 Actions available in the revPH phase

definition *readPrefOfPC* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *userID* ⇒ *preference*
where
readPrefOfPC s confID uID p papID uID′ ≡
pref s uID′ papID

definition *e-readPrefOfPC* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *userID*⇒ *bool*
where
e-readPrefOfPC s confID uID p papID uID′ ≡
IDsOK s [*confID*] [*uID*,*uID′*] [*papID*] ∧ *pass s uID* = *p* ∧
(*phase s confID* ≥ *bidPH* ∧ *isChair s confID uID* ∧ *isPC s confID uID′*
∨
phase s confID = *subPH* ∧ *isAut s confID uID papID*)

definition *createReview* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *userID* ⇒ *state*
where
createReview s confID uID p papID uID′ ≡
case paper s papID of Paper title abstract pc reviews dis decs ⇒
let rls = *roles s confID uID′*; *n* = *length* (*reviewsPaper* (*paper s papID*));
reviews′ = *reviews* @ [*emptyReview*]
in
s ⦇*roles* := *fun-upd2* (*roles s*) *confID uID′* (*List.insert* (*Rev papID n*) *rls*),
paper := *fun-upd* (*paper s*) *papID* (*Paper title abstract pc reviews′ dis decs*)
⦈

definition *e-createReview* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *userID* ⇒ *bool*
where
e-createReview s confID uID p papID uID′ ≡
IDsOK s [*confID*] [*uID*,*uID′*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID = *revPH* ∧
isChair s confID uID ∧ *pref s uID papID* ≠ *Conflict* ∧
isPC s confID uID′ ∧ ¬ *isRev s confID uID′ papID* ∧ *pref s uID′ papID* ≠ *Conflict*

definition *updateReview* ::
state ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *nat* ⇒ *rcontent* ⇒ *state*
where
updateReview s confID uID p papID n rc ≡
case paper s papID of Paper title abstract pc reviews dis decs ⇒
let review = [*rc*]; *reviews′* = *list-update reviews n review*
in
s ⦇*paper* := *fun-upd* (*paper s*) *papID* (*Paper title abstract pc reviews′ dis decs*)⦈

definition *e-updateReview* ::
state ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *nat* ⇒ *rcontent* ⇒ *bool*
where
e-updateReview s confID uID p papID n rc ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID = *revPH* ∧ *isRev s confID uID papID* ∧
getReviewIndex s confID uID papID = *n*

2.2.11 Actions available starting from the revPH phase

definition *readMyReview* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *nat* ∗ *review*
where
readMyReview s confID uID p papID ≡
case getRevRole s confID uID papID of
Some (*Rev papID′ n*) ⇒ (*n*, *getNthReview s papID n*)

definition *e-readMyReview* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*
where
e-readMyReview s confID uID p papID ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *revPH* ∧ *isRev s confID uID papID*

definition *listMyAssignedPapers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID list*
where
listMyAssignedPapers s confID uID p ≡
let paps = *paper s in*
[*papID*. *papID* ← *paperIDs s confID*, *isRev s confID uID papID*]

definition *e-listMyAssignedPapers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listMyAssignedPapers s confID uID p ≡
IDsOK s [*confID*] [*uID*] [] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *revPH* ∧ *isPC s confID uID*

definition *listAssignedReviewers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *userID list*
where
listAssignedReviewers s confID uID p papID ≡
[*uID* ← *userIDs s*. *isRev s confID uID papID*]

definition *e-listAssignedReviewers* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*

where
e-listAssignedReviewers s confID uID p papID $\equiv$
IDsOK s [*confID*] [*uID*] [*papID*] $\wedge$ *pass s uID* $=$ *p* $\wedge$
phase s confID $\geq$ *revPH* $\wedge$
isChair s confID uID $\wedge$ *pref s uID papID* $\neq$ *Conflict*

2.2.12 Actions available in the disPH phase

definition *uupdateDis* :: *state* $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *password* $\Rightarrow$ *paperID* $\Rightarrow$ *string* $\Rightarrow$ *state*
where
uupdateDis s confID uID p papID comm $\equiv$
case paper s papID of Paper title abstract pc reviews (*Dis comments*) *decs* $\Rightarrow$
s $(\!|$*paper* := *fun-upd* (*paper s*) *papID* (*Paper title abstract pc reviews* (*Dis* (*comm* # *comments*)) *decs*)$|\!)$

definition *e-uupdateDis* :: *state* $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *password* $\Rightarrow$ *paperID* $\Rightarrow$ *string* $\Rightarrow$ *bool*
where
e-uupdateDis s confID uID p papID comm $\equiv$
IDsOK s [*confID*] [*uID*] [*papID*] $\wedge$ *pass s uID* $=$ *p* $\wedge$
phase s confID $=$ *disPH* $\wedge$ *isPC s confID uID* $\wedge$ *pref s uID papID* $\neq$ *Conflict*

definition *uupdateReview* ::
state $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *password* $\Rightarrow$ *paperID* $\Rightarrow$ *nat* $\Rightarrow$ *rcontent* $\Rightarrow$ *state*
where
uupdateReview s confID uID p papID n rc $\equiv$
case paper s papID of Paper title abstract pc reviews dis decs $\Rightarrow$
let review $=$ *rc* # (*reviews* ! *n*); *reviews'* $=$ *list-update reviews n review*
in
s $(\!|$*paper* := *fun-upd* (*paper s*) *papID* (*Paper title abstract pc reviews' dis decs*)$|\!)$

definition *e-uupdateReview* ::
state $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *password* $\Rightarrow$ *paperID* $\Rightarrow$ *nat* $\Rightarrow$ *rcontent* $\Rightarrow$ *bool*
where
e-uupdateReview s confID uID p papID n rc $\equiv$
IDsOK s [*confID*] [*uID*] [*papID*] $\wedge$ *pass s uID* $=$ *p* $\wedge$
phase s confID $=$ *disPH* $\wedge$ *isRev s confID uID papID* $\wedge$
getReviewIndex s confID uID papID $=$ *n*

definition *uupdateDec* :: *state* $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *password* $\Rightarrow$ *paperID* $\Rightarrow$ *decision* $\Rightarrow$ *state*
where
uupdateDec s confID uID p papID dec $\equiv$
case paper s papID of Paper title abstract pc reviews dis decs $\Rightarrow$
s $(\!|$*paper* := *fun-upd* (*paper s*) *papID* (*Paper title abstract pc reviews dis* (*dec* # *decs*))$|\!)$

definition *e-uupdateDec* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *decision* ⇒ *bool*
where
e-uupdateDec s confID uID p papID dec ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID = *disPH* ∧ *isChair s confID uID* ∧ *pref s uID papID* ≠ *Conflict*

2.2.13 Actions available starting from the disPH phase

definition *readReviews* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ (*userID* ∗ *review*) *list*
where
readReviews s confID uID p papID ≡
getReviewersReviews s confID papID

definition *e-readReviews* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*
where
e-readReviews s confID uID p papID ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *disPH* ∧ *isPC s confID uID* ∧ *pref s uID papID* ≠ *Conflict*

definition *readDecs* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *decision list*
where
readDecs s confID uID p papID ≡
case paper s papID of Paper title abstract pc reviews dis decs ⇒ *decs*

definition *e-readDecs* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*
where
e-readDecs s confID uID p papID ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *disPH* ∧ *isPC s confID uID* ∧ *pref s uID papID* ≠ *Conflict*

definition *readDis* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *string list*
where
readDis s confID uID p papID ≡
case paper s papID of Paper title abstract pc reviews (*Dis comments*) *decs* ⇒ *comments*

definition *e-readDis* :: *state* ⇒ *confID* ⇒ *userID* ⇒ *password* ⇒ *paperID* ⇒ *bool*
where
e-readDis s confID uID p papID ≡
IDsOK s [*confID*] [*uID*] [*papID*] ∧ *pass s uID* = *p* ∧
phase s confID ≥ *disPH* ∧ *isPC s confID uID* ∧ *pref s uID papID* ≠ *Conflict*

2.2.14 Actions available starting from the notifPH phase

definition $readFinalReviews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow review\ list$
where
$readFinalReviews\ s\ confID\ uID\ p\ papID \equiv$
$map\ (\lambda\ rev.\ case\ rev\ of\ []\ \Rightarrow [(NoExp,NoScore,emptyStr)]$
$|((exp,score,comm)\ \#\ rv) \Rightarrow [(exp,score,comm)])$
$(reviewsPaper\ (paper\ s\ papID))$

definition $e\text{-}readFinalReviews :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$
where
$e\text{-}readFinalReviews\ s\ confID\ uID\ p\ papID \equiv$
$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$
$phase\ s\ confID \geq notifPH \wedge (isAut\ s\ confID\ uID\ papID \vee (isPC\ s\ confID\ uID \wedge pref\ s\ uID\ papID \neq Conflict))$

definition $readFinalDec :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow decision$
where
$readFinalDec\ s\ confID\ uID\ p\ papID \equiv$
$case\ paper\ s\ papID\ of\ Paper\ title\ abstract\ pc\ reviews\ dis\ decs \Rightarrow$
$case\ decs\ of\ [] \Rightarrow NoDecision \mid dec\ \#\ decs \Rightarrow dec$

definition $e\text{-}readFinalDec :: state \Rightarrow confID \Rightarrow userID \Rightarrow password \Rightarrow paperID \Rightarrow bool$
where
$e\text{-}readFinalDec\ s\ confID\ uID\ p\ papID \equiv$
$IDsOK\ s\ [confID]\ [uID]\ [papID] \wedge pass\ s\ uID = p \wedge$
$phase\ s\ confID \geq notifPH \wedge (isAut\ s\ confID\ uID\ papID \vee isPC\ s\ confID\ uID)$

2.3 The step function

datatype $out =$
$outOK \mid outErr \mid$
$outBool\ bool \mid$
$outSTRT\ string * string * string \mid outSTRL\ string\ list \mid outCONF\ string * string * role\ list * phase \mid$
$outPREF\ preference \mid$
$outCON\ pcontent \mid$
$outNREV\ nat * review \mid outREVL\ review\ list \mid outRREVL\ (userID * review)\ list \mid$
$outDEC\ decision \mid outDECL\ decision\ list \mid$
$outCIDL\ confID\ list \mid outUIDL\ userID\ list \mid outPIDL\ paperID\ list \mid$
$outSTRPAL\ string * string * userID\ list$

datatype $cAct =$
$cUser\ userID\ password\ string\ string\ string$

$|$*cConf confID userID password string string*
$|$*cPC confID userID password userID*
$|$*cChair confID userID password userID*
$|$*cPaper confID userID password paperID string string*
$|$*cAuthor confID userID password paperID userID*
$|$*cConflict confID userID password paperID userID*
$|$*cReview confID userID password paperID userID*

lemmas *c-defs* =
e-createUser-def createUser-def
e-createConf-def createConf-def
e-createPC-def createPC-def
e-createChair-def createChair-def
e-createAuthor-def createAuthor-def
e-createConflict-def createConflict-def
e-createPaper-def createPaper-def
e-createReview-def createReview-def

fun *cStep* :: *state* ⇒ *cAct* ⇒ *out* ∗ *state* **where**
cStep s (*cUser uID p name info email*) =
(*if e-createUser s uID p name info email*
then (*outOK*, *createUser s uID p name info email*)
else (*outErr*, *s*))
|
cStep s (*cConf confID uID p name info*) =
(*if e-createConf s confID uID p name info*
then (*outOK*, *createConf s confID uID p name info*)
else (*outErr*, *s*))
|
cStep s (*cPC confID uID p uID′*) =
(*if e-createPC s confID uID p uID′*
then (*outOK*, *createPC s confID uID p uID′*)
else (*outErr*, *s*))
|
cStep s (*cChair confID uID p uID′*) =
(*if e-createChair s confID uID p uID′*
then (*outOK*, *createChair s confID uID p uID′*)
else (*outErr*, *s*))
|
cStep s (*cPaper confID uID p papID name info*) =
(*if e-createPaper s confID uID p papID name info*
then (*outOK*, *createPaper s confID uID p papID name info*)
else (*outErr*, *s*))
|
cStep s (*cAuthor confID uID p papID uID′*) =
(*if e-createAuthor s confID uID p papID uID′*
then (*outOK*, *createAuthor s confID uID p papID uID′*)
else (*outErr*, *s*))
|

cStep s (*cConflict confID uID p papID uID'*) =
(*if e-createConflict s confID uID p papID uID'*
then (*outOK*, *createConflict s confID uID p papID uID'*)
else (*outErr*, *s*))
|
cStep s (*cReview confID uID p papID uID'*) =
(*if e-createReview s confID uID p papID uID'*
then (*outOK*, *createReview s confID uID p papID uID'*)
else (*outErr*, *s*))

datatype *uAct* =
uUser userID password password string string string
|*uConfA confID userID password*
|*uPhase confID userID password phase*
|*uPaperTA confID userID password paperID string string*
|*uPaperC confID userID password paperID pcontent*
|*uPref confID userID password paperID preference*
|*uReview confID userID password paperID nat rcontent*

lemmas *u-defs* =
e-updateUser-def updateUser-def
e-updateConfA-def updateConfA-def
e-updatePhase-def updatePhase-def
e-updatePaperTA-def updatePaperTA-def
e-updatePaperC-def updatePaperC-def
e-updatePref-def updatePref-def
e-updateReview-def updateReview-def

fun *uStep* :: *state* ⇒ *uAct* ⇒ *out* ∗ *state* **where**
uStep s (*uUser uID p p' name info email*) =
(*if e-updateUser s uID p p' name info email*
then (*outOK*, *updateUser s uID p p' name info email*)
else (*outErr*, *s*))
|
uStep s (*uConfA confID uID p*) =
(*if e-updateConfA s confID uID p*
then (*outOK*, *updateConfA s confID uID p*)
else (*outErr*, *s*))
|
uStep s (*uPhase confID uID p ph*) =
(*if e-updatePhase s confID uID p ph*
then (*outOK*, *updatePhase s confID uID p ph*)
else (*outErr*, *s*))
|
uStep s (*uPaperTA confID uID p papID name info*) =
(*if e-updatePaperTA s confID uID p papID name info*
then (*outOK*, *updatePaperTA s confID uID p papID name info*)
else (*outErr*, *s*))

|
uStep s (*uPaperC confID uID p papID pc*) =
(*if e-updatePaperC s confID uID p papID pc*
then (*outOK*, *updatePaperC s confID uID p papID pc*)
else (*outErr*, *s*))
|
uStep s (*uPref confID uID p papID pr*) =
(*if e-updatePref s confID uID p papID pr*
then (*outOK*, *updatePref s confID uID p papID pr*)
else (*outErr*, *s*))
|
uStep s (*uReview confID uID p papID n rc*) =
(*if e-updateReview s confID uID p papID n rc*
then (*outOK*, *updateReview s confID uID p papID n rc*)
else (*outErr*, *s*))

datatype *uuAct* =
uuNews confID userID password string
|*uuDis confID userID password paperID string*
|*uuReview confID userID password paperID nat rcontent*
|*uuDec confID userID password paperID decision*

lemmas *uu-defs* =
e-uupdateNews-def uupdateNews-def
e-uupdateDis-def uupdateDis-def
e-uupdateReview-def uupdateReview-def
uupdateDec-def e-uupdateDec-def

fun *uuStep* :: *state* ⇒ *uuAct* ⇒ *out* ∗ *state* **where**
uuStep s (*uuNews confID uID p ev*) =
(*if e-uupdateNews s confID uID p ev*
then (*outOK*, *uupdateNews s confID uID p ev*)
else (*outErr*, *s*))
|
uuStep s (*uuDis confID uID p papID comm*) =
(*if e-uupdateDis s confID uID p papID comm*
then (*outOK*, *uupdateDis s confID uID p papID comm*)
else (*outErr*, *s*))
|
uuStep s (*uuReview confID uID p papID n rc*) =
(*if e-uupdateReview s confID uID p papID n rc*
then (*outOK*, *uupdateReview s confID uID p papID n rc*)
else (*outErr*, *s*))
|
uuStep s (*uuDec confID uID p papID dec*) =
(*if e-uupdateDec s confID uID p papID dec*
then (*outOK*, *uupdateDec s confID uID p papID dec*)
else (*outErr*, *s*))

datatype *rAct* =
rAmIVoronkov userID password
|*rUser userID password userID*
|*rConf confID userID password*
|*rNews confID userID password*
|*rPaperTAA confID userID password paperID*
|*rPaperC confID userID password paperID*
|*rPref confID userID password paperID*
|*rMyReview confID userID password paperID*
|*rReviews confID userID password paperID*
|*rDecs confID userID password paperID*
|*rDis confID userID password paperID*
|*rFinalReviews confID userID password paperID*
|*rFinalDec confID userID password paperID*
|*rPrefOfPC confID userID password paperID userID*

lemmas *r-defs* =
readAmIVoronkov-def e-readAmIVoronkov-def
readUser-def e-readUser-def
readConf-def e-readConf-def
readNews-def e-readNews-def
readPaperTAA-def e-readPaperTAA-def
readPaperC-def e-readPaperC-def
readPref-def e-readPref-def
readMyReview-def e-readMyReview-def
readReviews-def e-readReviews-def
readDecs-def e-readDecs-def
readDis-def e-readDis-def
readFinalReviews-def e-readFinalReviews-def
readFinalDec-def e-readFinalDec-def
readPrefOfPC-def e-readPrefOfPC-def

fun *rObs* :: *state* ⇒ *rAct* ⇒ *out* **where**
rObs s (*rAmIVoronkov uID p*) =
(*if e-readAmIVoronkov s uID p then outBool* (*readAmIVoronkov s uID p*) *else outErr*)
|
rObs s (*rUser uID p uID'*) =
(*if e-readUser s uID p uID' then outSTRT* (*readUser s uID p uID'*) *else outErr*)
|
rObs s (*rConf confID uID p*) =
(*if e-readConf s confID uID p then outCONF* (*readConf s confID uID p*) *else outErr*)
|
rObs s (*rNews confID uID p*) =
(*if e-readNews s confID uID p then outSTRL* (*readNews s confID uID p*) *else outErr*)
|

rObs s (rPaperTAA confID uID p papID) =
(if e-readPaperTAA s confID uID p papID then outSTRPAL (readPaperTAA s confID uID p papID) else outErr)
|
rObs s (rPaperC confID uID p papID) =
(if e-readPaperC s confID uID p papID then outCON (readPaperC s confID uID p papID) else outErr)
|
rObs s (rPref confID uID p papID) =
(if e-readPref s confID uID p papID then outPREF (readPref s confID uID p papID) else outErr)
|
rObs s (rMyReview confID uID p papID) =
(if e-readMyReview s confID uID p papID then outNREV (readMyReview s confID uID p papID) else outErr)
|
rObs s (rReviews confID uID p papID) =
(if e-readReviews s confID uID p papID then outRREVL (readReviews s confID uID p papID) else outErr)
|
rObs s (rDecs confID uID p papID) =
(if e-readDecs s confID uID p papID then outDECL (readDecs s confID uID p papID) else outErr)
|
rObs s (rDis confID uID p papID) =
(if e-readDis s confID uID p papID then outSTRL (readDis s confID uID p papID) else outErr)
|
rObs s (rFinalReviews confID uID p papID) =
(if e-readFinalReviews s confID uID p papID then outREVL (readFinalReviews s confID uID p papID) else outErr)
|
rObs s (rFinalDec confID uID p papID) =
(if e-readFinalDec s confID uID p papID then outDEC (readFinalDec s confID uID p papID) else outErr)
|
rObs s (rPrefOfPC confID uID p papID uID′) =
(if e-readPrefOfPC s confID uID p papID uID′ then outPREF (readPrefOfPC s confID uID p papID uID′) else outErr)

datatype *lAct =*
lConfs userID password
|lAConfs userID password
|lSConfs userID password
|lMyConfs userID password
|lAllUsers userID password
|lAllPapers userID password
|lPC confID userID password
|lChair confID userID password

|*lPapers confID userID password*
|*lMyPapers confID userID password*
|*lMyAssignedPapers confID userID password*
|*lAssignedReviewers confID userID password paperID*

lemmas *l-defs* =
listConfs-def e-listConfs-def
listAConfs-def e-listAConfs-def
listSConfs-def e-listSConfs-def
listMyConfs-def e-listMyConfs-def
listAllUsers-def e-listAllUsers-def
listAllPapers-def e-listAllPapers-def
listPC-def e-listPC-def
listChair-def e-listChair-def
listPapers-def e-listPapers-def
listMyPapers-def e-listMyPapers-def
listMyAssignedPapers-def e-listMyAssignedPapers-def
listAssignedReviewers-def e-listAssignedReviewers-def

fun *lObs* :: *state* ⇒ *lAct* ⇒ *out* **where**
lObs s (*lConfs uID p*) =
(*if e-listConfs s uID p then outCIDL* (*listConfs s uID p*) *else outErr*)
|
lObs s (*lAConfs uID p*) =
(*if e-listAConfs s uID p then outCIDL* (*listAConfs s uID p*) *else outErr*)
|
lObs s (*lSConfs uID p*) =
(*if e-listSConfs s uID p then outCIDL* (*listSConfs s uID p*) *else outErr*)
|
lObs s (*lMyConfs uID p*) =
(*if e-listMyConfs s uID p then outCIDL* (*listMyConfs s uID p*) *else outErr*)
|
lObs s (*lAllUsers uID p*) =
(*if e-listAllUsers s uID p then outUIDL* (*listAllUsers s uID p*) *else outErr*)
|
lObs s (*lAllPapers uID p*) =
(*if e-listAllPapers s uID p then outPIDL* (*listAllPapers s uID p*) *else outErr*)
|
lObs s (*lPC confID uID p*) =
(*if e-listPC s confID uID p then outUIDL* (*listPC s confID uID p*) *else outErr*)
|
lObs s (*lChair confID uID p*) =
(*if e-listChair s confID uID p then outUIDL* (*listChair s confID uID p*) *else outErr*)
|
lObs s (*lPapers confID uID p*) =
(*if e-listPapers s confID uID p then outPIDL* (*listPapers s confID uID p*) *else outErr*)
|
lObs s (*lMyPapers confID uID p*) =

(*if e-listMyPapers s confID uID p then outPIDL* (*listMyPapers s confID uID p*) *else outErr*)
|
lObs s (*lMyAssignedPapers confID uID p*) =
(*if e-listMyAssignedPapers s confID uID p then outPIDL* (*listMyAssignedPapers s confID uID p*) *else outErr*)
|
lObs s (*lAssignedReviewers confID uID p papID*) =
(*if e-listAssignedReviewers s confID uID p papID*
then outUIDL (*listAssignedReviewers s confID uID p papID*) *else outErr*)

datatype *act* =

Cact cAct | *Uact uAct* | *UUact uuAct* |

Ract rAct | *Lact lAct*

fun *step* :: *state* ⇒ *act* ⇒ *out* ∗ *state* **where**
step s (*Cact ca*) = *cStep s ca*
|
step s (*Uact ua*) = *uStep s ua*
|
step s (*UUact uua*) = *uuStep s uua*
|
step s (*Ract ra*) = (*rObs s ra*, *s*)
|
step s (*Lact la*) = (*lObs s la*, *s*)

export-code *step istate getFreshPaperID* **in** *Scala*

Some action selectors, used for verification:

fun *cUserOfA* :: *cAct* ⇒ *userID* **where**
cUserOfA (*cUser uID p name info email*) = *uID*
|
cUserOfA (*cConf confID uID p name info*) = *uID*
|
cUserOfA (*cPC confID uID p uID'*) = *uID*
|
cUserOfA (*cChair confID uID p uID'*) = *uID*
|
cUserOfA (*cPaper confID uID p papID name info*) = *uID*
|
cUserOfA (*cAuthor confID uID p papID uID'*) = *uID*
|
cUserOfA (*cConflict confID uID p papID uID'*) = *uID*
|
cUserOfA (*cReview confID uID p papID uID'*) = *uID*

fun *uUserOfA* :: *uAct* ⇒ *userID* **where**

$uUserOfA\ (uUser\ uID\ p\ p'\ name\ info\ email) = uID$
|
$uUserOfA\ (uConfA\ confID\ uID\ p) = uID$
|
$uUserOfA\ (uPhase\ confID\ uID\ p\ ph) = uID$
|
$uUserOfA\ (uPaperTA\ confID\ uID\ p\ papID\ name\ info) = uID$
|
$uUserOfA\ (uPaperC\ confID\ uID\ p\ papID\ pc) = uID$
|
$uUserOfA\ (uPref\ confID\ uID\ p\ papID\ pr) = uID$
|
$uUserOfA\ (uReview\ confID\ uID\ p\ papID\ n\ rc) = uID$

fun $uuUserOfA :: uuAct \Rightarrow userID$ **where**
$uuUserOfA\ (uuNews\ confID\ uID\ p\ ev) = uID$
|
$uuUserOfA\ (uuDis\ confID\ uID\ p\ papID\ comm) = uID$
|
$uuUserOfA\ (uuReview\ confID\ uID\ p\ papID\ n\ rc) = uID$
|
$uuUserOfA\ (uuDec\ confID\ uID\ p\ papID\ dec) = uID$

fun $rUserOfA :: rAct \Rightarrow userID$ **where**
$rUserOfA\ (rAmIVoronkov\ uID\ p) = uID$
|
$rUserOfA\ (rUser\ uID\ p\ uID') = uID$
|
$rUserOfA\ (rConf\ confID\ uID\ p) = uID$
|
$rUserOfA\ (rNews\ confID\ uID\ p) = uID$
|
$rUserOfA\ (rPaperTAA\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rPaperC\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rPref\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rMyReview\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rReviews\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rDecs\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rDis\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rFinalReviews\ confID\ uID\ p\ papID) = uID$
|
$rUserOfA\ (rFinalDec\ confID\ uID\ p\ papID) = uID$

|
$rUserOfA\ (rPrefOfPC\ confID\ uID\ p\ papID\ uID') = uID$

fun $lUserOfA :: lAct \Rightarrow userID$ **where**
$lUserOfA\ (lConfs\ uID\ p) = uID$
|
$lUserOfA\ (lAConfs\ uID\ p) = uID$
|
$lUserOfA\ (lSConfs\ uID\ p) = uID$
|
$lUserOfA\ (lMyConfs\ uID\ p) = uID$
|
$lUserOfA\ (lAllUsers\ uID\ p) = uID$
|
$lUserOfA\ (lAllPapers\ uID\ p) = uID$
|
$lUserOfA\ (lPC\ confID\ uID\ p) = uID$
|
$lUserOfA\ (lChair\ confID\ uID\ p) = uID$
|
$lUserOfA\ (lPapers\ confID\ uID\ p) = uID$
|
$lUserOfA\ (lMyPapers\ confID\ uID\ p) = uID$
|
$lUserOfA\ (lMyAssignedPapers\ confID\ uID\ p) = uID$
|
$lUserOfA\ (lAssignedReviewers\ confID\ uID\ p\ papID) = uID$

fun $userOfA :: act \Rightarrow userID$ **where**
$userOfA\ (Cact\ ca) = cUserOfA\ ca$
|
$userOfA\ (Uact\ ua) = uUserOfA\ ua$
|
$userOfA\ (UUact\ uua) = uuUserOfA\ uua$
|
$userOfA\ (Ract\ ra) = rUserOfA\ ra$
|
$userOfA\ (Lact\ la) = lUserOfA\ la$

fun $cConfOfA :: cAct \Rightarrow confID\ option$ **where**
$cConfOfA\ (cUser\ uID\ p\ name\ info\ email) = None$
|
$cConfOfA\ (cConf\ confID\ uID\ p\ name\ info) = Some\ confID$

|
$cConfOfA\ (cPC\ confID\ uID\ p\ uID') = Some\ confID$
|

cConfOfA (*cChair confID uID p uID'*) = *Some confID*
|
cConfOfA (*cPaper confID uID p papID name info*) = *Some confID*
|
cConfOfA (*cAuthor confID uID p papID uID'*) = *Some confID*
|
cConfOfA (*cConflict confID uID p papID uID'*) = *Some confID*
|
cConfOfA (*cReview confID uID p papID uID'*) = *Some confID*

fun *uConfOfA* :: *uAct* ⇒ *confID option* **where**
uConfOfA (*uUser uID p p' name info email*) = *None*
|
uConfOfA (*uConfA confID uID p*) = *Some confID*
|
uConfOfA (*uPhase confID uID p ph*) = *Some confID*
|
uConfOfA (*uPaperTA confID uID p papID name info*) = *Some confID*
|
uConfOfA (*uPaperC confID uID p papID pc*) = *Some confID*
|
uConfOfA (*uPref confID uID p papID pr*) = *Some confID*
|
uConfOfA (*uReview confID uID p papID n rc*) = *Some confID*

fun *uuConfOfA* :: *uuAct* ⇒ *confID option* **where**
uuConfOfA (*uuNews confID uID p ev*) = *Some confID*
|
uuConfOfA (*uuDis confID uID p papID comm*) = *Some confID*
|
uuConfOfA (*uuReview confID uID p papID n rc*) = *Some confID*
|
uuConfOfA (*uuDec confID uID p papID dec*) = *Some confID*

fun *rConfOfA* :: *rAct* ⇒ *confID option* **where**
rConfOfA (*rAmIVoronkov uID p*) = *None*
|
rConfOfA (*rUser uID p uID'*) = *None*
|
rConfOfA (*rConf confID uID p*) = *Some confID*
|
rConfOfA (*rNews confID uID p*) = *Some confID*
|
rConfOfA (*rPaperTAA confID uID p papID*) = *Some confID*
|
rConfOfA (*rPaperC confID uID p papID*) = *Some confID*
|
rConfOfA (*rPref confID uID p papID*) = *Some confID*
|

$rConfOfA$ $(rMyReview$ $confID$ uID p $papID) = Some$ $confID$
|
$rConfOfA$ $(rReviews$ $confID$ uID p $papID) = Some$ $confID$
|
$rConfOfA$ $(rDecs$ $confID$ uID p $papID) = Some$ $confID$
|
$rConfOfA$ $(rDis$ $confID$ uID p $papID) = Some$ $confID$
|
$rConfOfA$ $(rFinalReviews$ $confID$ uID p $papID) = Some$ $confID$
|
$rConfOfA$ $(rFinalDec$ $confID$ uID p $papID) = Some$ $confID$
|
$rConfOfA$ $(rPrefOfPC$ $confID$ uID p $papID$ $uID') = Some$ $confID$

fun $lConfOfA :: lAct \Rightarrow confID$ $option$ **where**
$lConfOfA$ $(lConfs$ uID $p) = None$
|
$lConfOfA$ $(lAConfs$ uID $p) = None$
|
$lConfOfA$ $(lSConfs$ uID $p) = None$
|
$lConfOfA$ $(lMyConfs$ uID $p) = None$
|
$lConfOfA$ $(lAllUsers$ uID $p) = None$
|
$lConfOfA$ $(lAllPapers$ uID $p) = None$
|
$lConfOfA$ $(lPC$ $confID$ uID $p) = Some$ $confID$
|
$lConfOfA$ $(lChair$ $confID$ uID $p) = Some$ $confID$
|
$lConfOfA$ $(lPapers$ $confID$ uID $p) = Some$ $confID$
|
$lConfOfA$ $(lMyPapers$ $confID$ uID $p) = Some$ $confID$
|
$lConfOfA$ $(lMyAssignedPapers$ $confID$ uID $p) = Some$ $confID$
|
$lConfOfA$ $(lAssignedReviewers$ $confID$ uID p $papID) = Some$ $confID$

fun $confOfA :: act \Rightarrow confID$ $option$ **where**
$confOfA$ $(Cact$ $ca) = cConfOfA$ ca
|
$confOfA$ $(Uact$ $ua) = uConfOfA$ ua
|
$confOfA$ $(UUact$ $uua) = uuConfOfA$ uua
|
$confOfA$ $(Ract$ $ra) = rConfOfA$ ra
|

confOfA (*Lact la*) = *lConfOfA la*

fun *cPaperOfA* :: *cAct* ⇒ *paperID option* **where**
cPaperOfA (*cUser uID p name info email*) = *None*
|
cPaperOfA (*cPaper confID uID p papID name info*) = *Some papID*

|
cPaperOfA (*cPC confID uID p uID′*) = *None*
|
cPaperOfA (*cChair confID uID p uID′*) = *None*
|
cPaperOfA (*cConf confID uID p name info*) = *None*
|
cPaperOfA (*cAuthor confID uID p papID uID′*) = *Some papID*
|
cPaperOfA (*cConflict confID uID p papID uID′*) = *Some papID*
|
cPaperOfA (*cReview confID uID p papID uID′*) = *Some papID*

fun *uPaperOfA* :: *uAct* ⇒ *paperID option* **where**
uPaperOfA (*uUser uID p p′ name info email*) = *None*
|
uPaperOfA (*uConfA confID uID p*) = *None*
|
uPaperOfA (*uPhase confID uID p ph*) = *None*
|
uPaperOfA (*uPaperTA confID uID p papID name info*) = *Some papID*
|
uPaperOfA (*uPaperC confID uID p papID pc*) = *Some papID*
|
uPaperOfA (*uPref confID uID p papID pr*) = *Some papID*
|
uPaperOfA (*uReview confID uID p papID n rc*) = *Some papID*

fun *uuPaperOfA* :: *uuAct* ⇒ *paperID option* **where**
uuPaperOfA (*uuNews confID uID p ev*) = *None*
|
uuPaperOfA (*uuDis confID uID p papID comm*) = *Some papID*
|
uuPaperOfA (*uuReview confID uID p papID n rc*) = *Some papID*
|
uuPaperOfA (*uuDec confID uID p papID dec*) = *Some papID*

fun *rPaperOfA* :: *rAct* ⇒ *paperID option* **where**
rPaperOfA (*rAmIVoronkov uID p*) = *None*
|

$rPaperOfA\ (rUser\ uID\ p\ uID') = None$
|
$rPaperOfA\ (rConf\ confID\ uID\ p) = None$
|
$rPaperOfA\ (rNews\ confID\ uID\ p) = None$
|
$rPaperOfA\ (rPaperTAA\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rPaperC\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rPref\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rMyReview\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rReviews\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rDecs\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rDis\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rFinalReviews\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rFinalDec\ confID\ uID\ p\ papID) = Some\ papID$
|
$rPaperOfA\ (rPrefOfPC\ confID\ uID\ p\ papID\ uID') = Some\ papID$

fun $lPaperOfA :: lAct \Rightarrow paperID\ option$ **where**
$lPaperOfA\ (lConfs\ uID\ p) = None$
|
$lPaperOfA\ (lAConfs\ uID\ p) = None$
|
$lPaperOfA\ (lSConfs\ uID\ p) = None$
|
$lPaperOfA\ (lMyConfs\ uID\ p) = None$
|
$lPaperOfA\ (lAllUsers\ uID\ p) = None$
|
$lPaperOfA\ (lAllPapers\ uID\ p) = None$
|
$lPaperOfA\ (lPC\ confID\ uID\ p) = None$
|
$lPaperOfA\ (lChair\ confID\ uID\ p) = None$
|
$lPaperOfA\ (lPapers\ confID\ uID\ p) = None$
|
$lPaperOfA\ (lMyPapers\ confID\ uID\ p) = None$
|
$lPaperOfA\ (lMyAssignedPapers\ confID\ uID\ p) = None$
|

lPaperOfA (*lAssignedReviewers confID uID p papID*) = *Some papID*

fun *paperOfA* :: *act* ⇒ *paperID option* **where**
paperOfA (*Cact ca*) = *cPaperOfA ca*
|
paperOfA (*Uact ua*) = *uPaperOfA ua*
|
paperOfA (*UUact uua*) = *uuPaperOfA uua*
|
paperOfA (*Ract ra*) = *rPaperOfA ra*
|
paperOfA (*Lact la*) = *lPaperOfA la*

end
theory *Automation-Setup*
imports *System-Specification*
begin

lemma *add-prop*:
  assumes *PROP* (*T*)
  shows *A* ==> *PROP* (*T*)
  ⟨*proof*⟩

lemmas *exhaust-elim* =
   cAct.exhaust[*of x*, *THEN add-prop*[**where** *A*=*a*=*Cact x*], *rotated* −*1*]
   uAct.exhaust[*of x*, *THEN add-prop*[**where** *A*=*a*=*Uact x*], *rotated* −*1*]
   uuAct.exhaust[*of x*, *THEN add-prop*[**where** *A*=*a*=*UUact x*], *rotated* −*1*]
   rAct.exhaust[*of x*, *THEN add-prop*[**where** *A*=*a*=*Ract x*], *rotated* −*1*]
   lAct.exhaust[*of x*, *THEN add-prop*[**where** *A*=*a*=*Lact x*], *rotated* −*1*]
  for *x a*

lemma *Paper-dest-conv*:
  (*p* =
      Paper title abstract content reviews dis decs) ⟷
  title = *titlePaper p* ∧
  abstract = *abstractPaper p* ∧
  content = *contentPaper p* ∧
  reviews = *reviewsPaper p* ∧
  dis = *disPaper p* ∧
  decs = *decsPaper p*

  ⟨*proof*⟩

end

3 Safety properties

theory *Safety-Properties*
imports *Automation-Setup Bounded-Deducibility-Security.IO-Automaton*
begin

interpretation *IO-Automaton* **where**
istate = *istate* **and** *step* = *step*
$\langle proof \rangle$

3.1 Infrastructure for invariance reasoning

definition *cIsInvar* :: (*state* ⇒ *bool*) ⇒ *bool* **where**
cIsInvar $\varphi \equiv \forall$ *s ca. reach s* $\wedge$ φ *s* $\longrightarrow$ φ (*snd* (*cStep s ca*))

definition *uIsInvar* :: (*state* ⇒ *bool*) ⇒ *bool* **where**
uIsInvar $\varphi \equiv \forall$ *s ua. reach s* $\wedge$ φ *s* $\longrightarrow$ φ (*snd* (*uStep s ua*))

definition *uuIsInvar* :: (*state* ⇒ *bool*) ⇒ *bool* **where**
uuIsInvar $\varphi \equiv \forall$ *s uua. reach s* $\wedge$ φ *s* $\longrightarrow$ φ (*snd* (*uuStep s uua*))

lemma *invar-cIsInvar-uIsInvar-uuIsInvar*:
invar $\varphi \longleftrightarrow$ *cIsInvar* φ $\wedge$ *uIsInvar* φ $\wedge$ *uuIsInvar* φ (**is** *?L* $\longleftrightarrow$ *?R*)
$\langle proof \rangle$

lemma *cIsInvar*[*case-names cUser cConf cPC cChair cPaper cAuthor cConflict cReview*]:
assumes
$\bigwedge$*s uID p name info email.*
  $[\![$*reach s*; φ *s*; *e-createUser s uID p name info email*$]\!]$
  $\Longrightarrow \varphi$ (*createUser s uID p name info email*)
and
$\bigwedge$*s confID uID p name info.*
  $[\![$*reach s*; φ *s*; *e-createConf s confID uID p name info*$]\!]$
  $\Longrightarrow \varphi$ (*createConf s confID uID p name info*)
and
$\bigwedge$*s confID uID p uID'.*
  $[\![$*reach s*; φ *s*; *e-createPC s confID uID p uID'*$]\!]$
  $\Longrightarrow \varphi$ (*createPC s confID uID p uID'*)
and
$\bigwedge$*s confID uID p uID'.*
  $[\![$*reach s*; φ *s*; *e-createChair s confID uID p uID'*$]\!]$
  $\Longrightarrow \varphi$ (*createChair s confID uID p uID'*)
and
$\bigwedge$*s confID uID p papID name info.*

$[\![reach\ s;\ \varphi\ s;\ e\text{-}createPaper\ s\ confID\ uID\ p\ papID\ name\ info]\!]$
$\Longrightarrow \varphi\ (createPaper\ s\ confID\ uID\ p\ papID\ name\ info)$
and
$\bigwedge s\ confID\ uID\ p\ papID\ uID'.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}createAuthor\ s\ confID\ uID\ p\ papID\ uID']\!]$
$\Longrightarrow \varphi\ (createAuthor\ s\ confID\ uID\ p\ papID\ uID')$
and
$\bigwedge s\ confID\ uID\ p\ papID\ uID'.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}createConflict\ s\ confID\ uID\ p\ papID\ uID']\!]$
$\Longrightarrow \varphi\ (createConflict\ s\ confID\ uID\ p\ papID\ uID')$
and
$\bigwedge s\ confID\ uID\ p\ papID\ uID'.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}createReview\ s\ confID\ uID\ p\ papID\ uID']\!]$
$\Longrightarrow \varphi\ (createReview\ s\ confID\ uID\ p\ papID\ uID')$
shows *cIsInvar* φ
$\langle proof \rangle$

lemma *uIsInvar*[*case-names uUser uConfA uNextPhase uPaperTA uPaperC uPref uReview*]:
assumes
$\bigwedge s\ uID\ p\ p'\ name\ info\ email.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}updateUser\ s\ uID\ p\ p'\ name\ info\ email]\!]$
$\Longrightarrow \varphi\ (updateUser\ s\ uID\ p\ p'\ name\ info\ email)$
and
$\bigwedge s\ confID\ uID\ p.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}updateConfA\ s\ confID\ uID\ p]\!] \Longrightarrow \varphi\ (updateConfA\ s\ confID\ uID\ p)$
and
$\bigwedge s\ confID\ uID\ p\ ph.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}updatePhase\ s\ confID\ uID\ p\ ph]\!] \Longrightarrow \varphi\ (updatePhase\ s\ confID\ uID\ p\ ph)$
and
$\bigwedge s\ confID\ uID\ p\ paperID\ name\ info.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}updatePaperTA\ s\ confID\ uID\ p\ paperID\ name\ info]\!]$
$\Longrightarrow \varphi\ (updatePaperTA\ s\ confID\ uID\ p\ paperID\ name\ info)$
and
$\bigwedge s\ confID\ uID\ p\ paperID\ pc.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}updatePaperC\ s\ confID\ uID\ p\ paperID\ pc]\!]$
$\Longrightarrow \varphi\ (updatePaperC\ s\ confID\ uID\ p\ paperID\ pc)$
and
$\bigwedge s\ confID\ uID\ p\ paperID\ preference.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}updatePref\ s\ confID\ uID\ p\ paperID\ preference]\!]$
$\Longrightarrow \varphi\ (updatePref\ s\ confID\ uID\ p\ paperID\ preference)$
and
$\bigwedge s\ confID\ uID\ p\ paperID\ n\ rc.$
$[\![reach\ s;\ \varphi\ s;\ e\text{-}updateReview\ s\ confID\ uID\ p\ paperID\ n\ rc]\!]$
$\Longrightarrow \varphi\ (updateReview\ s\ confID\ uID\ p\ paperID\ n\ rc)$
and
$\bigwedge s\ confID\ uID\ p\ paperID\ fpc.$

$[\![$*reach s*; φ *s*; *e-updateFPaperC s confID uID p paperID fpc*$]\!]$
$\Longrightarrow \varphi$ (*updateFPaperC s confID uID p paperID fpc*)
shows *uIsInvar* φ
⟨*proof*⟩

lemma *uuIsInvar*[*case-names uuNews uuDis uuReview uuDec*]:
assumes
$\bigwedge$*s confID uID p comm.*
$[\![$*reach s*; φ *s*; *e-uupdateNews s confID uID p comm*$]\!]$
$\Longrightarrow \varphi$ (*uupdateNews s confID uID p comm*)
and
$\bigwedge$*s confID uID p paperID comm.*
$[\![$*reach s*; φ *s*; *e-uupdateDis s confID uID p paperID comm*$]\!]$
$\Longrightarrow \varphi$ (*uupdateDis s confID uID p paperID comm*)
and
$\bigwedge$*s confID uID p paperID n rc.*
$[\![$*reach s*; φ *s*; *e-uupdateReview s confID uID p paperID n rc*$]\!]$
$\Longrightarrow \varphi$ (*uupdateReview s confID uID p paperID n rc*)
and
$\bigwedge$*s confID uID p paperID decision.*
$[\![$*reach s*; φ *s*; *e-uupdateDec s confID uID p paperID decision*$]\!]$
$\Longrightarrow \varphi$ (*uupdateDec s confID uID p paperID decision*)
shows *uuIsInvar* φ
⟨*proof*⟩

3.2 Safety proofs

declare *option.splits*[*split*] *paper.splits*[*split*] *discussion.splits*[*split*] *role.splits*[*split*]
Let-def[*simp*] *list-all-iff*[*simp*] *list-ex-iff*[*simp*] *fun-upd2-def*[*simp*] *IDsOK-def*[*simp*]
if-splits[*split*]

fun *papIDsOfRole* **where**
papIDsOfRole (*Aut papID*) = [*papID*]
|
papIDsOfRole (*Rev papID n*) = [*papID*]
|
papIDsOfRole - = []

definition *phase-leq-closedPH* :: *state* ⇒ *bool* **where**
phase-leq-closedPH s ≡
∀ *confID. phase s confID* ≤ *closedPH*

lemma *holdsIstate-phase-leq-closedPH*: *holdsIstate phase-leq-closedPH*
⟨*proof*⟩

lemma *cIsInvar-phase-leq-closedPH*: *cIsInvar phase-leq-closedPH*
⟨*proof*⟩

lemma *uIsInvar-phase-leq-closedPH*: *uIsInvar phase-leq-closedPH*
⟨*proof*⟩

lemma *uuIsInvar-phase-leq-closedPH*: *uuIsInvar phase-leq-closedPH*
⟨*proof*⟩

lemma *invar-phase-leq-closedPH*: *invar phase-leq-closedPH*
⟨*proof*⟩

lemmas *phase-leq-closedPH1* =
holdsIstate-invar[*OF holdsIstate-phase-leq-closedPH invar-phase-leq-closedPH*]

theorem *phase-leq-closedPH*:
assumes *a*: *reach s*
shows *phase s confID* ≤ *closedPH*
⟨*proof*⟩

definition *geq-noPH-confIDs* :: *state* ⇒ *bool* **where**
geq-noPH-confIDs s ≡
∀ *confID*. *phase s confID* > *noPH* ⟶ *confID* ∈∈ *confIDs s*

lemma *holdsIstate-geq-noPH-confIDs*: *holdsIstate geq-noPH-confIDs*
⟨*proof*⟩

lemma *cIsInvar-geq-noPH-confIDs*: *cIsInvar geq-noPH-confIDs*
⟨*proof*⟩

lemma *uIsInvar-geq-noPH-confIDs*: *uIsInvar geq-noPH-confIDs*
⟨*proof*⟩

lemma *uuIsInvar-geq-noPH-confIDs*: *uuIsInvar geq-noPH-confIDs*
⟨*proof*⟩

lemma *invar-geq-noPH-confIDs*: *invar geq-noPH-confIDs*
⟨*proof*⟩

lemmas *geq-noPH-confIDs1* =
holdsIstate-invar[*OF holdsIstate-geq-noPH-confIDs invar-geq-noPH-confIDs*]

theorem *geq-noPH-confIDs*:
assumes *a*: *reach s*
shows *phase s confID* > *noPH* ⟶ *confID* ∈∈ *confIDs s*
⟨*proof*⟩

definition *roles-IDsOK* :: *state* ⇒ *bool* **where**
roles-IDsOK s ≡

$\forall$ *confID uID rl.*
rl $\in\in$ *roles s confID uID* $\longrightarrow$ *IDsOK s* [*confID*] [*uID*] (*papIDsOfRole rl*)

lemma *holdsIstate-roles-IDsOK*: *holdsIstate roles-IDsOK*
⟨*proof*⟩

lemma *cIsInvar-roles-IDsOK*: *cIsInvar roles-IDsOK*
⟨*proof*⟩

lemma *uIsInvar-roles-IDsOK*: *uIsInvar roles-IDsOK*
⟨*proof*⟩

lemma *uuIsInvar-roles-IDsOK*: *uuIsInvar roles-IDsOK*
⟨*proof*⟩

lemma *invar-roles-IDsOK*: *invar roles-IDsOK*
⟨*proof*⟩

lemmas *roles-IDsOK1* =
holdsIstate-invar[*OF holdsIstate-roles-IDsOK invar-roles-IDsOK*]

theorem *roles-IDsOK*:
assumes *a*: *reach s* **and** *rl*: *rl* $\in\in$ *roles s confID uID*
shows *IDsOK s* [*confID*] [*uID*] (*papIDsOfRole rl*)
⟨*proof*⟩

corollary *roles-confIDs*:
assumes *a*: *reach s* **and** *A*: *rl* $\in\in$ *roles s confID uID*
shows *confID* $\in\in$ *confIDs s*
⟨*proof*⟩

corollary *roles-userIDs*:
assumes *a*: *reach s* **and** *A*: *rl* $\in\in$ *roles s confID uID*
shows *uID* $\in\in$ *userIDs s*
⟨*proof*⟩

corollary *isAut-paperIDs*:
assumes *a*: *reach s* **and** *A*: *isAut s confID uID papID*
shows *papID* $\in\in$ *paperIDs s confID*
⟨*proof*⟩

corollary *isRevNth-paperIDs*:
assumes *a*: *reach s* **and** *A*: *isRevNth s confID uID papID n*
shows *papID* $\in\in$ *paperIDs s confID*
⟨*proof*⟩

corollary *isRev-paperIDs*:
assumes *a*: *reach s* **and** *A*: *isRev s confID uID papID*
shows *papID* $\in\in$ *paperIDs s confID*

⟨*proof*⟩

corollary *isRev-userIDs*:
assumes *a*: *reach s* **and** *A*: *isRev s confID uID papID*
shows *uID* ∈∈ *userIDs s*
⟨*proof*⟩

corollary *isRev-confIDs*:
assumes *a*: *reach s* **and** *A*: *isRev s confID uID papID*
shows *confID* ∈∈ *confIDs s*
⟨*proof*⟩

definition *distinct-IDs* :: *state* ⇒ *bool* **where**
distinct-IDs s ≡
distinct (*confIDs s*) ∧ *distinct* (*userIDs s*) ∧ (∀ *confID*. *distinct* (*paperIDs s confID*))

lemma *holdsIstate-distinct-IDs*: *holdsIstate distinct-IDs*
⟨*proof*⟩

lemma *cIsInvar-distinct-IDs*: *cIsInvar distinct-IDs*
⟨*proof*⟩

lemma *uIsInvar-distinct-IDs*: *uIsInvar distinct-IDs*
⟨*proof*⟩

lemma *uuIsInvar-distinct-IDs*: *uuIsInvar distinct-IDs*
⟨*proof*⟩

lemma *invar-distinct-IDs*: *invar distinct-IDs*
⟨*proof*⟩

lemmas *distinct-IDs1* = *holdsIstate-invar*[*OF holdsIstate-distinct-IDs invar-distinct-IDs*]

theorem *distinct-IDs*:
assumes *a*: *reach s*
shows *distinct* (*confIDs s*) ∧ *distinct* (*userIDs s*) ∧ (∀ *confID*. *distinct* (*paperIDs s confID*))
⟨*proof*⟩

lemmas *distinct-confIDs* = *distinct-IDs*[*THEN conjunct1*]
lemmas *distinct-userIDs* = *distinct-IDs*[*THEN conjunct2*, *THEN conjunct1*]
lemmas *distinct-paperIDs* = *distinct-IDs*[*THEN conjunct2*, *THEN conjunct2*, *rule-format*]

definition *distinct-roles* :: *state* ⇒ *bool* **where**
distinct-roles s ≡
∀ *confID uID*. *distinct* (*roles s confID uID*)

lemma *holdsIstate-distinct-roles*: *holdsIstate distinct-roles*
⟨*proof*⟩

lemma *cIsInvar-distinct-roles*: *cIsInvar distinct-roles*
⟨*proof*⟩

lemma *uIsInvar-distinct-roles*: *uIsInvar distinct-roles*
⟨*proof*⟩

lemma *uuIsInvar-distinct-roles*: *uuIsInvar distinct-roles*
⟨*proof*⟩

lemma *invar-distinct-roles*: *invar distinct-roles*
⟨*proof*⟩

lemmas *distinct-roles1* = *holdsIstate-invar*[*OF holdsIstate-distinct-roles invar-distinct-roles*]

theorem *distinct-roles*:
assumes *a*: *reach s*
shows *distinct* (*roles s confID uID*)
⟨*proof*⟩

definition *isRevNth-isPC* :: *state* ⇒ *bool* **where**
isRevNth-isPC s ≡
∀ *confID uID papID n*. *isRevNth s confID uID papID n* ⟶ *isPC s confID uID*

lemma *holdsIstate-isRevNth-isPC*: *holdsIstate isRevNth-isPC*
⟨*proof*⟩

lemma *cIsInvar-isRevNth-isPC*: *cIsInvar isRevNth-isPC*
⟨*proof*⟩

lemma *uIsInvar-isRevNth-isPC*: *uIsInvar isRevNth-isPC*
⟨*proof*⟩

lemma *uuIsInvar-isRevNth-isPC*: *uuIsInvar isRevNth-isPC*
⟨*proof*⟩

lemma *invar-isRevNth-isPC*: *invar isRevNth-isPC*
⟨*proof*⟩

lemmas *isRevNth-isPC1* = *holdsIstate-invar*[*OF holdsIstate-isRevNth-isPC invar-isRevNth-isPC*]

theorem *isRevNth-isPC*:
assumes *a*: *reach s* **and** *R*: *isRevNth s confID uID papID n*
shows *isPC s confID uID*
⟨*proof*⟩

corollary *isRev-isPC*:
assumes *a*: *reach s* **and** *R*: *isRev s confID uID papID*
shows *isPC s confID uID*
⟨*proof*⟩

definition *paperIDs-confIDs* :: *state* ⇒ *bool* **where**
paperIDs-confIDs s ≡
∀ *confID papID*.
papID ∈∈ *paperIDs s confID* ⟶ *confID* ∈∈ *confIDs s*

lemma *holdsIstate-paperIDs-confIDs*: *holdsIstate paperIDs-confIDs*
⟨*proof*⟩

lemma *cIsInvar-paperIDs-confIDs*: *cIsInvar paperIDs-confIDs*
⟨*proof*⟩

lemma *uIsInvar-paperIDs-confIDs*: *uIsInvar paperIDs-confIDs*
⟨*proof*⟩

lemma *uuIsInvar-paperIDs-confIDs*: *uuIsInvar paperIDs-confIDs*
⟨*proof*⟩

lemma *invar-paperIDs-confIDs*: *invar paperIDs-confIDs*
⟨*proof*⟩

lemmas *paperIDs-confIDs1* = *holdsIstate-invar*[*OF holdsIstate-paperIDs-confIDs invar-paperIDs-confIDs*]

theorem *paperIDs-confIDs*:
assumes *a*: *reach s* **and** *p*: *papID* ∈∈ *paperIDs s confID*
shows *confID* ∈∈ *confIDs s*
⟨*proof*⟩

corollary *paperIDs-getAllPaperIDs*:
assumes *a*: *reach s* **and** *p*: *papID* ∈∈ *paperIDs s confID*
shows *papID* ∈∈ *getAllPaperIDs s*
⟨*proof*⟩

corollary *isRevNth-getAllPaperIDs*:
assumes *a*: *reach s* **and** *isRevNth s confID uID papID n*
shows *papID* ∈∈ *getAllPaperIDs s*
⟨*proof*⟩

definition *paperIDs-equals* :: *state* ⇒ *bool* **where**
paperIDs-equals s ≡
∀ *confID1 confID2 papID*.

papID ∈∈ *paperIDs s confID1* ∧ *papID* ∈∈ *paperIDs s confID2*
⟶ *confID1* = *confID2*

lemma *holdsIstate-paperIDs-equals*: *holdsIstate paperIDs-equals*
⟨*proof*⟩

lemma *cIsInvar-paperIDs-equals*: *cIsInvar paperIDs-equals*
⟨*proof*⟩

lemma *uIsInvar-paperIDs-equals*: *uIsInvar paperIDs-equals*
⟨*proof*⟩

lemma *uuIsInvar-paperIDs-equals*: *uuIsInvar paperIDs-equals*
⟨*proof*⟩

lemma *invar-paperIDs-equals*: *invar paperIDs-equals*
⟨*proof*⟩

lemmas *paperIDs-equals1* = *holdsIstate-invar*[*OF holdsIstate-paperIDs-equals invar-paperIDs-equals*]

theorem *paperIDs-equals*:
assumes *a*: *reach s* **and** *p*: *papID* ∈∈ *paperIDs s confID1 papID* ∈∈ *paperIDs s confID2*
shows *confID1* = *confID2*
⟨*proof*⟩

definition *isAut-pref-Conflict* :: *state* ⇒ *bool* **where**
isAut-pref-Conflict s ≡
∀ *confID uID papID*. *isAut s confID uID papID* ⟶ *pref s uID papID* = *Conflict*

lemma *holdsIstate-isAut-pref-Conflict*: *holdsIstate isAut-pref-Conflict*
⟨*proof*⟩

lemma *cIsInvar-isAut-pref-Conflict*: *cIsInvar isAut-pref-Conflict*
⟨*proof*⟩

lemma *uIsInvar-isAut-pref-Conflict*: *uIsInvar isAut-pref-Conflict*
⟨*proof*⟩

lemma *uuIsInvar-isAut-pref-Conflict*: *uuIsInvar isAut-pref-Conflict*
⟨*proof*⟩

lemma *invar-isAut-pref-Conflict*: *invar isAut-pref-Conflict*
⟨*proof*⟩

lemmas *isAut-pref-Conflict1* =
holdsIstate-invar[*OF holdsIstate-isAut-pref-Conflict invar-isAut-pref-Conflict*]

theorem *isAut-pref-Conflict*:
assumes *a*: *reach s* **and** *i*: *isAut s confID uID papID*
shows *pref s uID papID* = *Conflict*
⟨*proof*⟩

definition *phase-noPH-paperIDs* :: *state* ⇒ *bool* **where**
phase-noPH-paperIDs s ≡
∀ *confID*. *phase s confID* = *noPH* ⟶ *paperIDs s confID* = []

lemma *holdsIstate-phase-noPH-paperIDs*: *holdsIstate phase-noPH-paperIDs*
⟨*proof*⟩

lemma *cIsInvar-phase-noPH-paperIDs*: *cIsInvar phase-noPH-paperIDs*
⟨*proof*⟩

lemma *uIsInvar-phase-noPH-paperIDs*: *uIsInvar phase-noPH-paperIDs*
⟨*proof*⟩

lemma *uuIsInvar-phase-noPH-paperIDs*: *uuIsInvar phase-noPH-paperIDs*
⟨*proof*⟩

lemma *invar-phase-noPH-paperIDs*: *invar phase-noPH-paperIDs*
⟨*proof*⟩

lemmas *phase-noPH-paperIDs1* =
holdsIstate-invar[*OF holdsIstate-phase-noPH-paperIDs invar-phase-noPH-paperIDs*]

theorem *phase-noPH-paperIDs*:
assumes *a*: *reach s* **and** *p*: *phase s confID* = *noPH*
shows *paperIDs s confID* = []
⟨*proof*⟩

definition *paperIDs-geq-subPH* :: *state* ⇒ *bool* **where**
paperIDs-geq-subPH s ≡
∀ *confID papID*. *papID* ∈∈ *paperIDs s confID* ⟶ *phase s confID* ≥ *subPH*

lemma *holdsIstate-paperIDs-geq-subPH*: *holdsIstate paperIDs-geq-subPH*
⟨*proof*⟩

lemma *cIsInvar-paperIDs-geq-subPH*: *cIsInvar paperIDs-geq-subPH*
⟨*proof*⟩

lemma *uIsInvar-paperIDs-geq-subPH*: *uIsInvar paperIDs-geq-subPH*
⟨*proof*⟩

lemma *uuIsInvar-paperIDs-geq-subPH*: *uuIsInvar paperIDs-geq-subPH*

⟨proof⟩

lemma *invar-paperIDs-geq-subPH*: *invar paperIDs-geq-subPH*
⟨proof⟩

lemmas *paperIDs-geq-subPH1* =
holdsIstate-invar[*OF holdsIstate-paperIDs-geq-subPH invar-paperIDs-geq-subPH*]

theorem *paperIDs-geq-subPH*:
assumes *a*: *reach s* **and** *i*: *papID* ∈∈ *paperIDs s confID*
shows *phase s confID* ≥ *subPH*
⟨proof⟩

definition *isRevNth-geq-revPH* :: *state* ⇒ *bool* **where**
isRevNth-geq-revPH s ≡
∀ *confID uID papID n*. *isRevNth s confID uID papID n* ⟶ *phase s confID* ≥ *revPH*

lemma *holdsIstate-isRevNth-geq-revPH*: *holdsIstate isRevNth-geq-revPH*
⟨proof⟩

lemma *cIsInvar-isRevNth-geq-revPH*: *cIsInvar isRevNth-geq-revPH*
⟨proof⟩

lemma *uIsInvar-isRevNth-geq-revPH*: *uIsInvar isRevNth-geq-revPH*
⟨proof⟩

lemma *uuIsInvar-isRevNth-geq-revPH*: *uuIsInvar isRevNth-geq-revPH*
⟨proof⟩

lemma *invar-isRevNth-geq-revPH*: *invar isRevNth-geq-revPH*
⟨proof⟩

lemmas *isRevNth-geq-revPH1* =
holdsIstate-invar[*OF holdsIstate-isRevNth-geq-revPH invar-isRevNth-geq-revPH*]

theorem *isRevNth-geq-revPH*:
assumes *a*: *reach s* **and** *i*: *isRevNth s confID uID papID n*
shows *phase s confID* ≥ *revPH*
⟨proof⟩

corollary *isRev-geq-revPH*:
assumes *a*: *reach s* **and** *i*: *isRev s confID uID papID*
shows *phase s confID* ≥ *revPH*
⟨proof⟩

definition *paperID-ex-userID* :: *state* ⇒ *bool* **where**

paperID-ex-userID s ≡
∀ *confID papID. papID* ∈∈ *paperIDs s confID* ⟶ (∃ *uID. isAut s confID uID papID*)

lemma *holdsIstate-paperID-ex-userID*: *holdsIstate paperID-ex-userID*
⟨*proof*⟩

lemma *cIsInvar-paperID-ex-userID*: *cIsInvar paperID-ex-userID*
⟨*proof*⟩

lemma *uIsInvar-paperID-ex-userID*: *uIsInvar paperID-ex-userID*
⟨*proof*⟩

lemma *uuIsInvar-paperID-ex-userID*: *uuIsInvar paperID-ex-userID*
⟨*proof*⟩

lemma *invar-paperID-ex-userID*: *invar paperID-ex-userID*
⟨*proof*⟩

lemmas *paperID-ex-userID1* =
holdsIstate-invar[*OF holdsIstate-paperID-ex-userID invar-paperID-ex-userID*]

theorem *paperID-ex-userID*:
assumes *a*: *reach s* **and** *i*: *papID* ∈∈ *paperIDs s confID*
shows ∃ *uID. isAut s confID uID papID*
⟨*proof*⟩

definition *pref-Conflict-isRevNth* :: *state* ⇒ *bool* **where**
pref-Conflict-isRevNth s ≡
∀ *confID uID papID n. pref s uID papID* = *Conflict* ⟶ ¬ *isRevNth s confID uID papID n*

lemma *holdsIstate-pref-Conflict-isRevNth*: *holdsIstate pref-Conflict-isRevNth*
⟨*proof*⟩

lemma *cIsInvar-pref-Conflict-isRevNth*: *cIsInvar pref-Conflict-isRevNth*
⟨*proof*⟩

lemma *uIsInvar-pref-Conflict-isRevNth*: *uIsInvar pref-Conflict-isRevNth*
⟨*proof*⟩

lemma *uuIsInvar-pref-Conflict-isRevNth*: *uuIsInvar pref-Conflict-isRevNth*
⟨*proof*⟩

lemma *invar-pref-Conflict-isRevNth*: *invar pref-Conflict-isRevNth*
⟨*proof*⟩

lemmas *pref-Conflict-isRevNth1* =

holdsIstate-invar[*OF holdsIstate-pref-Conflict-isRevNth invar-pref-Conflict-isRevNth*]

theorem *pref-Conflict-isRevNth*:
assumes *a*: *reach s* **and** *i*: *pref s uID papID* = *Conflict*
shows ¬ *isRevNth s confID uID papID n*
⟨*proof*⟩

corollary *pref-Conflict-isRev*:
assumes *a*: *reach s* **and** *i*: *pref s uID papID* = *Conflict*
shows ¬ *isRev s confID uID papID*
⟨*proof*⟩

corollary *pref-isAut-isRevNth*:
assumes *a*: *reach s* **and** *i*: *isAut s confID uID papID*
shows ¬ *isRevNth s confID uID papID n*
⟨*proof*⟩

corollary *pref-isAut-isRev*:
assumes *a*: *reach s* **and** *i*: *isAut s confID uID papID*
shows ¬ *isRev s confID uID papID*
⟨*proof*⟩

definition *isChair-isPC* :: *state* ⇒ *bool* **where**
isChair-isPC s ≡
∀ *confID uID*. *isChair s confID uID* ⟶ *isPC s confID uID*

lemma *holdsIstate-isChair-isPC*: *holdsIstate isChair-isPC*
⟨*proof*⟩

lemma *cIsInvar-isChair-isPC*: *cIsInvar isChair-isPC*
⟨*proof*⟩

lemma *uIsInvar-isChair-isPC*: *uIsInvar isChair-isPC*
⟨*proof*⟩

lemma *uuIsInvar-isChair-isPC*: *uuIsInvar isChair-isPC*
⟨*proof*⟩

lemma *invar-isChair-isPC*: *invar isChair-isPC*
⟨*proof*⟩

lemmas *isChair-isPC1* =
holdsIstate-invar[*OF holdsIstate-isChair-isPC invar-isChair-isPC*]

theorem *isChair-isPC*:
assumes *a*: *reach s* **and** *p*: *isChair s confID uID*
shows *isPC s confID uID*

⟨*proof*⟩

definition *isRevNth-equals* :: *state* ⇒ *bool* **where**
isRevNth-equals s ≡
∀ *confID uID papID m n*.
isRevNth s confID uID papID m ∧ *isRevNth s confID uID papID n*
⟶ *m* = *n*

lemma *holdsIstate-isRevNth-equals*: *holdsIstate isRevNth-equals*
⟨*proof*⟩

lemma *cIsInvar-isRevNth-equals*: *cIsInvar isRevNth-equals*
⟨*proof*⟩

lemma *uIsInvar-isRevNth-equals*: *uIsInvar isRevNth-equals*
⟨*proof*⟩

lemma *uuIsInvar-isRevNth-equals*: *uuIsInvar isRevNth-equals*
⟨*proof*⟩

lemma *invar-isRevNth-equals*: *invar isRevNth-equals*
⟨*proof*⟩

lemmas *isRevNth-equals1* =
holdsIstate-invar[*OF holdsIstate-isRevNth-equals invar-isRevNth-equals*]

theorem *isRevNth-equals*:
assumes *a*: *reach s* **and** *r*: *isRevNth s confID uID papID m isRevNth s confID uID papID n*
shows *m* = *n*
⟨*proof*⟩

corollary *isRevNth-getReviewIndex*:
assumes *a*: *reach s* **and** *r*: *isRevNth s confID uID papID n*
shows *n* = *getReviewIndex s confID uID papID*
⟨*proof*⟩

definition *isRevNth-less-length* :: *state* ⇒ *bool* **where**
isRevNth-less-length s ≡
∀ *confID uID papID n*.
isRevNth s confID uID papID n ⟶ *n* < *length* (*reviewsPaper* (*paper s papID*))

lemma *holdsIstate-isRevNth-less-length*: *holdsIstate isRevNth-less-length*
⟨*proof*⟩

lemma *cIsInvar-isRevNth-less-length*: *cIsInvar isRevNth-less-length*

⟨*proof*⟩

lemma *uIsInvar-isRevNth-less-length*: *uIsInvar isRevNth-less-length*
⟨*proof*⟩

lemma *uuIsInvar-isRevNth-less-length*: *uuIsInvar isRevNth-less-length*
⟨*proof*⟩

lemma *invar-isRevNth-less-length*: *invar isRevNth-less-length*
⟨*proof*⟩

lemmas *isRevNth-less-length1* =
holdsIstate-invar[*OF holdsIstate-isRevNth-less-length invar-isRevNth-less-length*]

theorem *isRevNth-less-length*:
assumes *reach s* **and** *isRevNth s cid uid pid n*
shows $n <$ *length* (*reviewsPaper* (*paper s pid*))
⟨*proof*⟩

definition *isRevNth-equalsU* :: *state* ⇒ *bool* **where**
isRevNth-equalsU s ≡
∀ *confID uID uID1 papID n*.
isRevNth s confID uID papID n ∧ *isRevNth s confID uID1 papID n*
⟶ *uID* = *uID1*

lemma *holdsIstate-isRevNth-equalsU*: *holdsIstate isRevNth-equalsU*
⟨*proof*⟩

lemma *cIsInvar-isRevNth-equalsU*: *cIsInvar isRevNth-equalsU*
⟨*proof*⟩

lemma *uIsInvar-isRevNth-equalsU*: *uIsInvar isRevNth-equalsU*
⟨*proof*⟩

lemma *uuIsInvar-isRevNth-equalsU*: *uuIsInvar isRevNth-equalsU*
⟨*proof*⟩

lemma *invar-isRevNth-equalsU*: *invar isRevNth-equalsU*
⟨*proof*⟩

lemmas *isRevNth-equalsU1* =
holdsIstate-invar[*OF holdsIstate-isRevNth-equalsU invar-isRevNth-equalsU*]

theorem *isRevNth-equalsU*:
assumes *a*: *reach s* **and** *r*: *isRevNth s confID uID papID n isRevNth s confID uID1 papID n*
shows *uID* = *uID1*

⟨*proof*⟩

definition *reviews-compact* :: *state* ⇒ *bool* **where**
reviews-compact s ≡
∀ *confID papID n*.
papID ∈∈ *paperIDs s confID* ∧ *n* < *length* (*reviewsPaper* (*paper s papID*)) ⟶
(∃ *uID*. *isRevNth s confID uID papID n*)

lemma *holdsIstate-reviews-compact*: *holdsIstate reviews-compact*
⟨*proof*⟩

lemma *cIsInvar-reviews-compact*: *cIsInvar reviews-compact*
⟨*proof*⟩

lemma *uIsInvar-reviews-compact*: *uIsInvar reviews-compact*
⟨*proof*⟩

lemma *uuIsInvar-reviews-compact*: *uuIsInvar reviews-compact*
⟨*proof*⟩

lemma *invar-reviews-compact*: *invar reviews-compact*
⟨*proof*⟩

lemmas *reviews-compact1* =
holdsIstate-invar[*OF holdsIstate-reviews-compact invar-reviews-compact*]

theorem *reviews-compact*:
assumes *reach s* **and** *n* < *length* (*reviewsPaper* (*paper s pid*))
and *pid* ∈∈ *paperIDs s cid*
shows ∃ *uid*. *isRevNth s cid uid pid n*
⟨*proof*⟩

definition *roles-nonrep* :: *state* ⇒ *bool* **where**
roles-nonrep s ≡
∀ *confID uID*.
distinct (*roles s confID uID*)

lemma *holdsIstate-roles-nonrep*: *holdsIstate roles-nonrep*
⟨*proof*⟩

lemma *cIsInvar-roles-nonrep*: *cIsInvar roles-nonrep*
⟨*proof*⟩

lemma *uIsInvar-roles-nonrep*: *uIsInvar roles-nonrep*
⟨*proof*⟩

lemma *uuIsInvar-roles-nonrep*: *uuIsInvar roles-nonrep*
⟨*proof*⟩

lemma *invar-roles-nonrep*: *invar roles-nonrep*
⟨*proof*⟩

lemmas *roles-nonrep1* =
holdsIstate-invar[*OF holdsIstate-roles-nonrep invar-roles-nonrep*]

theorem *roles-nonrep*:
assumes *reach s*
shows *distinct* (*roles s confID uID*)
⟨*proof*⟩

3.3 Properties of the step function

lemma *step-outErr-eq*: *step s a* = (*outErr*, s') $\Longrightarrow$ s'= *s*
⟨*proof*⟩

lemma *phase-increases*:
assumes *step s a* = (*ou*,s')
shows *phase s cid* $\leq$ *phase* s' *cid*
⟨*proof*⟩

lemma *phase-increases2*: *phase s CID* $\leq$ *phase* (*snd* (*step s a*)) *CID*
⟨*proof*⟩

lemma *confIDs-mono*:
assumes *step s a* = (*ou*,s') **and** *cid* ∈∈ *confIDs s*
shows *cid* ∈∈ *confIDs* s'
⟨*proof*⟩

lemma *userIDs-mono*:
assumes *step s a* = (*ou*,s') **and** *uid* ∈∈ *userIDs s*
shows *uid* ∈∈ *userIDs* s'
⟨*proof*⟩

lemma *paperIDs-mono*:
assumes *step s a* = (*ou*,s') **and** *pid* ∈∈ *paperIDs s cid*
shows *pid* ∈∈ *paperIDs* s' *cid*
⟨*proof*⟩

lemma *isPC-persistent*:
assumes *isPC s cid uid* **and** *step s a* = (*ou*, s')
shows *isPC* s' *cid uid*
⟨*proof*⟩

lemma *isChair-persistent*:
assumes *isChair s cid uid* **and** *step s a* = (*ou*, s')

shows *isChair* s' *cid uid*
⟨*proof*⟩

3.4 Action-safety properties

lemma *pref-Conflict-disPH*:
assumes *reach s* **and** *pid* $\in\in$ *paperIDs s cid* **and** *pref s uid pid* $\neq$ *Conflict* **and** *phase s cid* = *disPH*
and *step s a* = (*ou*, s')
shows *pref* s' *uid pid* $\neq$ *Conflict*
⟨*proof*⟩

lemma *isRevNth-persistent*:
assumes *reach s* **and** *isRevNth s cid uid pid n*
and *step s a* = (*ou*, s')
shows *isRevNth* s' *cid uid pid n*
⟨*proof*⟩

lemma *nonempty-decsPaper-persist*:
assumes *s*: *reach s*
and *pid*: *pid* $\in\in$ *paperIDs s cid*
and *decsPaper* (*paper s pid*) $\neq$ [] **and** *step s a* = (*ou*,s')
shows *decsPaper* (*paper* s' *pid*) $\neq$ []
⟨*proof*⟩

lemma *nonempty-reviews-persist*:
assumes *s*: *reach s*
and *r*: *isRevNth s cid uid pid n*
and (*reviewsPaper* (*paper s pid*))!*n* $\neq$ [] **and** *step s a* = (*ou*,s')
shows (*reviewsPaper* (*paper* s' *pid*))!*n* $\neq$ []
⟨*proof*⟩

lemma *revPH-pref-persists*:
assumes *reach s*
pid $\in\in$ *paperIDs s cid* **and** *phase s cid* $\geq$ *revPH*
and *step s a* = (*ou*,s')
shows *pref* s' *uid pid* = *pref s uid pid*
⟨*proof*⟩

3.5 Miscellaneous

lemma *updates-commute-paper*:
$\bigwedge$ *uu*. *s* (|*confIDs* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *confIDs* := *uu*|)
$\bigwedge$ *uu*. *s* (|*conf* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *conf* := *uu*|)

$\bigwedge$ *uu*. *s* (|*userIDs* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *userIDs* := *uu*|)
$\bigwedge$ *uu*. *s* (|*pass* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *pass* := *uu*|)
$\bigwedge$ *uu*. *s* (|*user* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *user* := *uu*|)
$\bigwedge$ *uu*. *s* (|*roles* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *roles* := *uu*|)

$\bigwedge$ *uu. s* (|*paperIDs* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *paperIDs* := *uu*|)

$\bigwedge$ *uu. s* (|*pref* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *pref* := *uu*|)
$\bigwedge$ *uu. s* (|*voronkov* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *voronkov* := *uu*|)
$\bigwedge$ *uu. s* (|*news* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *news* := *uu*|)
$\bigwedge$ *uu. s* (|*phase* := *uu*, *paper* := *pp*|) = *s* (|*paper* := *pp*, *phase* := *uu*|)
⟨*proof*⟩

lemma *isAUT-imp-isAut*:
assumes *reach s* **and** *pid* ∈∈ *paperIDs s cid* **and** *isAUT s uid pid*
shows *isAut s cid uid pid*
⟨*proof*⟩

lemma *isREVNth-imp-isRevNth*:
assumes *reach s* **and** *pid* ∈∈ *paperIDs s cid* **and** *isREVNth s uid pid n*
shows *isRevNth s cid uid pid n*
⟨*proof*⟩

lemma *phase-increases-validTrans*:
assumes *validTrans* (*Trans s a ou s'*)
shows *phase s cid* ≤ *phase s' cid*
⟨*proof*⟩

lemma *phase-increases-validTrans2*:
assumes *validTrans tr*
shows *phase* (*srcOf tr*) *cid* ≤ *phase* (*tgtOf tr*) *cid*
⟨*proof*⟩

lemma *phase-increases-trace*:
assumes *vtr*: *valid tr* **and** *ij*: *i* ≤ *j* **and** *j*: *j* < *length tr*
shows *phase* (*srcOf* (*tr*!*i*)) *cid* ≤ *phase* (*srcOf* (*tr*!*j*)) *cid*
⟨*proof*⟩

lemma *phase-increases-trace-srcOf-tgtOf*:
assumes *vtr*: *valid tr* **and** *ij*: *i* ≤ *j* **and** *j*: *j* < *length tr*
shows *phase* (*srcOf* (*tr*!*i*)) *cid* ≤ *phase* (*tgtOf* (*tr*!*j*)) *cid*
⟨*proof*⟩

lemma *phase-increases-trace-srcOf-hd*:
assumes *v*: *valid tr* **and** *l*: *length tr* > *1* **and** *i* < *length tr*
shows *phase* (*srcOf* (*hd tr*)) *cid* ≤ *phase* (*srcOf* (*tr*!*i*)) *cid*
⟨*proof*⟩

lemma *phase-increases-trace-srcOf-last*:
assumes *v*: *valid tr* **and** *l*: *length tr* $>$ *1* **and** *i*: *i* $<$ *length tr*
shows *phase* (*srcOf* (*tr*!*i*)) *cid* $\leq$ *phase* (*srcOf* (*last tr*)) *cid*
⟨*proof*⟩

lemma *phase-increases-trace-srcOf-tgtOf-last*:
assumes *v*: *valid tr* **and** *l*: *length tr* $>$ *1* **and** *i*: *i* $<$ *length tr*
shows *phase* (*srcOf* (*tr*!*i*)) *cid* $\leq$ *phase* (*tgtOf* (*last tr*)) *cid*
⟨*proof*⟩

lemma *valid-tgtPf-last-srcOf*:
assumes *valid tr* **and** *s* ∈∈ *map tgtOf tr*
shows *s* = *tgtOf* (*last tr*) ∨ *s* ∈∈ *map srcOf tr*
⟨*proof*⟩

lemma *phase-constant*:
assumes *v*: *valid tr* **and** *l*: *length tr* $>$ *0* **and**
ph: *phase* (*srcOf* (*hd tr*)) *cid* = *phase* (*tgtOf* (*last tr*)) *cid*
shows *set* (*map* (λ *trn*. *phase* (*srcOf trn*) *cid*) *tr*) ⊆ {*phase* (*srcOf* (*hd tr*)) *cid*}
∧
set (*map* (λ *trn*. *phase* (*tgtOf trn*) *cid*) *tr*) ⊆ {*phase* (*srcOf* (*hd tr*)) *cid*}
⟨*proof*⟩

lemma *phase-cases*:
assumes *step s a* = (*ou*, *s*′)
obtains (*noPH*) ¬ *cid* ∈∈ *confIDs s* ∨ *phase s cid* = *noPH*

| (*Id*) *phase s*′ *cid* = *phase s cid*
| (*Upd*) *uid p ph* **where** *phase s*′ *cid* = *ph a* = *Uact* (*uPhase cid uid p ph*)
e-updatePhase s cid uid p ph
⟨*proof*⟩

lemma *phase-mono*: *reachFrom s s*′ ⟹ *phase s cid* $\leq$ *phase s*′ *cid*
⟨*proof*⟩

lemma *validTrans-rAct-lAct-srcOf-tgtOf*:
assumes *validTrans trn*
and *actOf trn* = *Ract rAct* ∨ *actOf trn* = *Lact lAct*
shows *tgtOf trn* = *srcOf trn*
⟨*proof*⟩

lemma *valid-rAct-lAct-srcOf-tgtOf*:
assumes *valid tr*
and ⋀ *a*. *a* ∈∈ *map actOf tr* ⟹ (∃ *rAct*. *a* = *Ract rAct*) ∨ (∃ *lAct*. *a* = *Lact lAct*)
shows *srcOf* ‘ (*set tr*) ⊆ {*srcOf* (*hd tr*)}
⟨*proof*⟩

lemma *validFrom-rAct-lAct-srcOf-tgtOf*:

assumes *validFrom s tr*
and $\bigwedge$ *a. a* $\in\in$ *map actOf tr* $\Longrightarrow$ ($\exists$ *rAct. a* = *Ract rAct*) $\vee$ ($\exists$ *lAct. a* = *Lact lAct*)
shows *srcOf* ‘ (*set tr*) $\subseteq$ {*s*}
⟨*proof*⟩

lemma *tgtOf-last-traceOf-Ract-Lact*[*simp*]:
assumes *al* $\neq$ [] *set al* $\subseteq$ *range Ract* $\cup$ *range Lact*
shows *tgtOf* (*last* (*traceOf s al*)) = *s*
⟨*proof*⟩

lemma *paperIDs-cases*:
assumes *step s a* = (*ou*, *s'*)
obtains (*Id*) *paperIDs s' cid* = *paperIDs s cid*
| (*Create*) *cid uid p pid tit ab* **where**
paperIDs s' cid = *pid* # *paperIDs s cid a* = *Cact* (*cPaper cid uid p pid tit ab*)
e-createPaper s cid uid p pid tit ab
⟨*proof*⟩

lemma *paperIDs-decPH-const*:
assumes *s*: *step s a* = (*ou*, *s'*) **and** *phase s cid* > *subPH*
shows *paperIDs s' cid* = *paperIDs s cid*
⟨*proof*⟩

end
theory *Observation-Setup*
imports *Safety-Properties*
begin

4 Observation setup for confidentiality properties

The observation infrastructure, consisting of a discriminator γ and a selector g, is the same for all our confidentiality properties. Namely, we fix a group UIDs of users, and consider the actions and outputs of these users.

consts *UIDs* :: *userID set*

type-synonym *obs* = *act* $*$ *out*

fun γ :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *bool* **where**
γ (*Trans* - *a* - -) = (*userOfA a* $\in$ *UIDs*)

fun *g* :: (*state*,*act*,*out*)*trans* $\Rightarrow$ *obs* **where**
g (*Trans* - *a ou* -) = (*a*,*ou*)

end
theory *Paper-Intro*
imports *../Safety-Properties*
begin

5 Paper Confidentiality

In this section, we prove confidentiality properties for the papers submitted to a conference. The secrets (values) of interest are therefore the different versions of a given paper (with identifier PID) uploaded into the system.

The two properties that we prove represent points of "compromise" between the strength of the declassification bound and that of the declassification trigger. Let

- T1 denote "the paper's authorship"
- T2 denote "PC membership and the conference having reached the bidding phase"

The two bound-trigger combinations are:

- weak trigger (T1 or T2) paired with strong bound (nothing can be learned, save for some harmless information, namely the non-existence of any upload);
- strong trigger (T1) paired with weak bound (allowing to learn the last submitted version of the paper (but nothing more than that)).

end

theory *Paper-Value-Setup*
imports *Paper-Intro*
begin

consts *PID* :: *paperID*

5.1 Preliminaries

declare *updates-commute-paper*[*simp*]

fun *eqButC* :: *paper* ⇒ *paper* ⇒ *bool* **where**
eqButC (*Paper name info ct reviews dis decs*)
(*Paper name1 info1 ct1 reviews1 dis1 decs1*) =
(*name* = *name1* ∧ *info* = *info1* ∧ *reviews* = *reviews1* ∧ *dis* = *dis1* ∧ *decs* = *decs1*)

lemma *eqButC*:
eqButC pap pap1 =
(*titlePaper pap* = *titlePaper pap1* ∧ *abstractPaper pap* = *abstractPaper pap1* ∧
reviewsPaper pap = *reviewsPaper pap1* ∧ *disPaper pap* = *disPaper pap1* ∧ *decsPaper pap* = *decsPaper pap1*)
⟨*proof*⟩

lemma *eqButC-eq*[*simp*,*intro*!]: *eqButC pap pap*
⟨*proof*⟩

lemma *eqButC-sym*:
assumes *eqButC pap pap1*
shows *eqButC pap1 pap*
⟨*proof*⟩

lemma *eqButC-trans*:
assumes *eqButC pap pap1* **and** *eqButC pap1 pap2*
shows *eqButC pap pap2*
⟨*proof*⟩

definition *eeqButPID* **where**
eeqButPID paps paps1 ≡
∀ *pid*. *if pid* = *PID then eqButC* (*paps pid*) (*paps1 pid*) *else paps pid* = *paps1 pid*

lemma *eeqButPID-eeq*[*simp*,*intro*!]: *eeqButPID s s*
⟨*proof*⟩

lemma *eeqButPID-sym*:
assumes *eeqButPID s s1* **shows** *eeqButPID s1 s*
⟨*proof*⟩

lemma *eeqButPID-trans*:
assumes *eeqButPID s s1* **and** *eeqButPID s1 s2* **shows** *eeqButPID s s2*
⟨*proof*⟩

lemma *eeqButPID-imp*:
eeqButPID paps paps1 ⟹ *eqButC* (*paps PID*) (*paps1 PID*)
⟦*eeqButPID paps paps1*; *pid* ≠ *PID*⟧ ⟹ *paps pid* = *paps1 pid*
⟨*proof*⟩

lemma *eeqButPID-cong*:
assumes *eeqButPID paps paps1*
and *pid* = *PID* ⟹ *eqButC uu uu1*
and *pid* ≠ *PID* ⟹ *uu* = *uu1*
shows *eeqButPID* (*paps* (*pid* := *uu*)) (*paps1*(*pid* := *uu1*))
⟨*proof*⟩

lemma *eeqButPID-RDD*:
eeqButPID paps paps1 $\Longrightarrow$
titlePaper (*paps PID*) = *titlePaper* (*paps1 PID*) $\wedge$
abstractPaper (*paps PID*) = *abstractPaper* (*paps1 PID*) $\wedge$
reviewsPaper (*paps PID*) = *reviewsPaper* (*paps1 PID*) $\wedge$
disPaper (*paps PID*) = *disPaper* (*paps1 PID*) $\wedge$
decsPaper (*paps PID*) = *decsPaper* (*paps1 PID*)
⟨*proof*⟩

definition *eqButPID* :: *state* ⇒ *state* ⇒ *bool* **where**
eqButPID s s1 ≡
confIDs s = *confIDs s1* $\wedge$ *conf s* = *conf s1* $\wedge$
userIDs s = *userIDs s1* $\wedge$ *pass s* = *pass s1* $\wedge$ *user s* = *user s1* $\wedge$ *roles s* = *roles s1* $\wedge$
paperIDs s = *paperIDs s1*
$\wedge$
eeqButPID (*paper s*) (*paper s1*)
$\wedge$
pref s = *pref s1* $\wedge$
voronkov s = *voronkov s1* $\wedge$
news s = *news s1* $\wedge$ *phase s* = *phase s1*

lemma *eqButPID-eq*[*simp*,*intro*!]: *eqButPID s s*
⟨*proof*⟩

lemma *eqButPID-sym*:
assumes *eqButPID s s1* **shows** *eqButPID s1 s*
⟨*proof*⟩

lemma *eqButPID-trans*:
assumes *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*
⟨*proof*⟩

lemma *eqButPID-imp*:
eqButPID s s1 $\Longrightarrow$
confIDs s = *confIDs s1* $\wedge$ *conf s* = *conf s1* $\wedge$
userIDs s = *userIDs s1* $\wedge$ *pass s* = *pass s1* $\wedge$ *user s* = *user s1* $\wedge$ *roles s* = *roles s1* $\wedge$
paperIDs s = *paperIDs s1*
$\wedge$
eeqButPID (*paper s*) (*paper s1*)
$\wedge$
pref s = *pref s1* $\wedge$
voronkov s = *voronkov s1* $\wedge$
news s = *news s1* $\wedge$ *phase s* = *phase s1* $\wedge$

getAllPaperIDs s = *getAllPaperIDs s1* $\wedge$

$isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid\ \wedge$
$getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid\ \wedge$
$getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
$\langle proof \rangle$

lemma *eqButPID-imp1*:
$eqButPID\ s\ s1 \Longrightarrow eqButC\ (paper\ s\ pid)\ (paper\ s1\ pid)$
$eqButPID\ s\ s1 \Longrightarrow pid \neq PID \vee PID \neq pid \Longrightarrow$
$paper\ s\ pid = paper\ s1\ pid\ \wedge$
$getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
$\langle proof \rangle$

lemma *eqButPID-imp2*:
assumes $eqButPID\ s\ s1$ **and** $pid \neq PID \vee PID \neq pid$
shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
$\langle proof \rangle$

lemma *eqButPID-RDD*:
$eqButPID\ s\ s1 \Longrightarrow$
$titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID)\ \wedge$
$abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID)\ \wedge$
$reviewsPaper\ (paper\ s\ PID) = reviewsPaper\ (paper\ s1\ PID)\ \wedge$
$disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID)\ \wedge$
$decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$
$\langle proof \rangle$

lemma *eqButPID-cong*[*simp*, *intro*]:
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|confIDs := uu1|))\ (s1\ (|confIDs := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|conf := uu1|))\ (s1\ (|conf := uu2|))$

$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|userIDs := uu1|))\ (s1\ (|userIDs := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|pass := uu1|))\ (s1\ (|pass := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|user := uu1|))\ (s1\ (|user := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|roles := uu1|))\ (s1\ (|roles := uu2|))$

$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|paperIDs := uu1|))\ (s1\ (|paperIDs := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow eeqButPID\ uu1\ uu2 \Longrightarrow eqButPID\ (s\ (|paper := uu1|))\ (s1\ (|paper := uu2|))$

$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|pref := uu1|))\ (s1\ (|pref := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|voronkov := uu1|))$

$(s1\ (|voronkov := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|news := uu1|))\ (s1\ (|news := uu2|))$
$\bigwedge uu1\ uu2.\ eqButPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButPID\ (s\ (|phase := uu1|))\ (s1\ (|phase := uu2|))$

$\langle proof \rangle$

lemma *eqButPID-Paper*:
assumes *s's1'*: *eqButPID s s1*
and *paper s pid = Paper title abstract pc reviews dis decs*
and *paper s1 pid = Paper title1 abstract1 pc1 reviews1 dis1 decs1*
shows $title = title1 \wedge abstract = abstract1 \wedge reviews = reviews1 \wedge dis = dis1 \wedge decs = decs1$
$\langle proof \rangle$

definition $NOSIMP\ a \equiv a$
lemma [*cong*]: $NOSIMP\ a = NOSIMP\ a$ $\langle proof \rangle$

lemma *eqButPID-paper*:
  assumes *eqButPID s s1*
  shows *paper s = (paper s1)(PID :=*
    Paper (titlePaper (paper s1 PID))
      (abstractPaper (paper s1 PID))
      (contentPaper (NOSIMP (paper s PID)))
      (reviewsPaper (paper s1 PID))
      (disPaper (paper s1 PID))
      (decsPaper (paper s1 PID))
)
  $\langle proof \rangle$

lemmas *eqButPID-simps = eqButPID-imp eqButPID-paper*

5.2 Value Setup

type-synonym *value = pcontent*

fun φ :: *(state,act,out) trans* $\Rightarrow$ *bool* **where**
φ *(Trans - (Uact (uPaperC cid uid p pid ct)) ou -) = (pid = PID* $\wedge$ *ou = outOK)*
|
φ *- = False*

lemma φ*-def2*:
$\varphi\ (Trans\ s\ a\ ou\ s') = (\exists\ cid\ uid\ p\ ct.\ a = Uact\ (uPaperC\ cid\ uid\ p\ PID\ ct) \wedge ou = outOK)$
$\langle proof \rangle$

fun *f* :: *(state,act,out) trans* $\Rightarrow$ *value* **where**

f (*Trans* - (*Uact* (*uPaperC cid uid p pid ct*)) - -) = *ct*

lemma *Uact-uPaperC-step-eqButPID*:
assumes *a*: *a* = *Uact* (*uPaperC cid uid p PID ct*)
and *step s a* = (*ou*,*s'*)
shows *eqButPID s s'*
⟨*proof*⟩

lemma *φ-step-eqButPID*:
assumes *φ*: *φ* (*Trans s a ou s'*)
and *s*: *step s a* = (*ou*,*s'*)
shows *eqButPID s s'*
⟨*proof*⟩

lemma *eqButPID-step*:
assumes *s's1'*: *eqButPID s s1*
and *step*: *step s a* = (*ou*,*s'*)
and *step1*: *step s1 a* = (*ou1*,*s1'*)
shows *eqButPID s' s1'*
⟨*proof*⟩

lemma *eqButPID-step-φ-imp*:
assumes *s's1'*: *eqButPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *φ*: *φ* (*Trans s a ou s'*)
shows *φ* (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqButPID-step-φ*:
assumes *s's1'*: *eqButPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
shows *φ* (*Trans s a ou s'*) = *φ* (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

end
theory *Paper-Aut-PC*
imports ../*Observation-Setup Paper-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*
begin

5.3 Confidentiality protection from users who are not the paper's authors or PC members

We verify the following property:

A group of users UIDs learns nothing about the various uploads of a paper PID (save for the non-existence of any upload) unless/until one of the following occurs:

- a user in UIDs becomes the paper's author or
- a user in UIDs becomes a PC member in the paper's conference and the conference moves to the bidding phase.

fun T :: $(state,act,out)$ $trans \Rightarrow bool$ **where**
$T\ (Trans\ \text{-}\ \text{-}\ ou\ s') =$
$(\exists\ uid \in UIDs.$
$\quad isAUT\ s'\ uid\ PID \vee$
$\quad (\exists\ cid.\ PID \in\in paperIDs\ s'\ cid \wedge isPC\ s'\ cid\ uid \wedge phase\ s'\ cid \geq bidPH)$
$)$

declare $T.simps$ $[simp\ del]$

definition B :: $value\ list \Rightarrow value\ list \Rightarrow bool$ **where**
$B\ vl\ vl1 \equiv vl \neq []$

interpretation *BD-Security-IO* **where**
$istate = istate$ **and** $step = step$ **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma *reachNT-non-isAut-isPC-isChair*:
assumes $reachNT\ s$ **and** $uid \in UIDs$
shows
$\neg\ isAut\ s\ cid\ uid\ PID\ \wedge$
$(isPC\ s\ cid\ uid \longrightarrow \neg\ PID \in\in paperIDs\ s\ cid \vee phase\ s\ cid \leq subPH) \wedge$
$(isChair\ s\ cid\ uid \longrightarrow \neg\ PID \in\in paperIDs\ s\ cid \vee phase\ s\ cid \leq subPH)$
$\langle proof \rangle$

lemma $P\text{-}\varphi\text{-}\gamma$:
assumes 1: $reachNT\ s$ **and** 2: $step\ s\ a = (ou,s')\ \varphi\ (Trans\ s\ a\ ou\ s')$
shows $\neg\ \gamma\ (Trans\ s\ a\ ou\ s')$
$\langle proof \rangle$

major

lemma *eqButPID-step-out*:
assumes $s's1'$: $eqButPID\ s\ s1$
and $step$: $step\ s\ a = (ou,s')$ **and** $step1$: $step\ s1\ a = (ou1,s1')$
and sT: $reachNT\ s$ **and** $s1$: $reach\ s1$
and PID: $PID \in\in paperIDs\ s\ cid$
and $UIDs$: $userOfA\ a \in UIDs$
shows $ou = ou1$
$\langle proof \rangle$

definition $\Delta 1$ :: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 1\ s\ vl\ s1\ vl1 \equiv \neg\ (\exists\ cid.\ PID \in\in paperIDs\ s\ cid) \wedge s = s1 \wedge B\ vl\ vl1$

definition $\Delta 2$ :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$\Delta 2$ *s vl s1 vl1* $\equiv$
$\exists$ *cid. PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* = *subPH* $\wedge$ *eqButPID s s1*

definition $\Delta 3$ :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$\Delta 3$ *s vl s1 vl1* $\equiv$
$\exists$ *cid. PID* $\in\in$ *paperIDs s cid* $\wedge$ *eqButPID s s1* $\wedge$ *phase s cid* > *subPH* $\wedge$ *vl*=[] $\wedge$
vl1=[]

definition Δe :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δe *s vl s1 vl1* $\equiv$
$\exists$ *cid. PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* > *subPH* $\wedge$ *vl* $\neq$ []

lemma *istate*-$\Delta 1$:
assumes *B*: *B vl vl1*
shows $\Delta 1$ *istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont*-$\Delta 1$: *unwind-cont* $\Delta 1$ {$\Delta 1$,$\Delta 2$,Δe}
⟨*proof*⟩

lemma *unwind-cont*-$\Delta 2$: *unwind-cont* $\Delta 2$ {$\Delta 2$,$\Delta 3$,Δe}
⟨*proof*⟩

lemma *unwind-cont*-$\Delta 3$: *unwind-cont* $\Delta 3$ {$\Delta 3$,Δe}
⟨*proof*⟩

definition *K1exit* **where**
K1exit s $\equiv$ $\exists$ *cid. phase s cid* > *subPH* $\wedge$ *PID* $\in\in$ *paperIDs s cid*

lemma *invarNT-K1exit*: *invarNT K1exit*
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal K1exit v*
⟨*proof*⟩

lemma *unwind-exit*-Δe: *unwind-exit* Δe
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Paper-Aut*
imports ../*Observation-Setup Paper-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*
begin

5.4 Confidentiality protection from non-authors

We verify the following property:

A group of users UIDs learns nothing about the various uploads of a paper PID except for the last (most recent) upload unless/until a user in UIDs becomes an author of the paper.

fun T :: $(state,act,out)$ $trans \Rightarrow bool$ **where**
T $(Trans$ - - ou $s') = (\exists$ $uid \in UIDs.$ $isAUT$ s' uid $PID)$

declare $T.simps$ $[simp$ $del]$

definition B :: $value$ $list \Rightarrow value$ $list \Rightarrow bool$ **where**
B vl $vl1$ $\equiv vl \neq [] \wedge vl1 \neq [] \wedge last$ $vl = last$ $vl1$

interpretation *BD-Security-IO* **where**
$istate = istate$ **and** $step = step$ **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma *reachNT-non-isAut*:
assumes *reachNT s* **and** $uid \in UIDs$
shows $\neg$ *isAut s cid uid PID*
$\langle proof \rangle$

lemma T-φ-γ:
assumes *1*: *reachNT s* **and** *2*: $step$ s $a = (ou,s')$ φ $(Trans$ s a ou $s')$
shows $\neg$ γ $(Trans$ s a ou $s')$
$\langle proof \rangle$

lemma *eqButPID-step-out*:
assumes *ss1*: *eqButPID s s1*
and *step*: $step$ s $a = (ou,s')$ **and** *step1*: $step$ $s1$ $a = (ou1,s1')$
and *sT*: *reachNT s* **and** *s1*: *reach s1*
and *PID*: $PID \in\in paperIDs$ s cid
and *ph*: $phase$ s $cid = subPH$
and *UIDs*: $userOfA$ $a \in UIDs$
shows $ou = ou1$
$\langle proof \rangle$

major

lemma *eqButPID-step-eq*:
assumes *ss1*: *eqButPID s s1*
and [*simp*]: $a{=}Uact$ $(uPaperC$ cid uid p PID $ct)$ $ou{=}outOK$
and *step*: $step$ s $a = (ou,$ $s')$ **and** *step1*: $step$ $s1$ $a = (ou',$ $s1')$

shows $s' = s1'$
⟨*proof*⟩

definition $\Delta 1$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 1$ *s vl s1 vl1* ≡
¬ (∃ *cid*. *PID* ∈∈ *paperIDs s cid*) ∧ *s* = *s1* ∧ *B vl vl1*

definition $\Delta 2$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 2$ *s vl s1 vl1* ≡
(∃ *cid*. *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* = *subPH*) ∧
eqButPID s s1 ∧ *B vl vl1*

definition $\Delta 3$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 3$ *s vl s1 vl1* ≡
(∃ *cid*. *PID* ∈∈ *paperIDs s cid*) ∧ *s* = *s1* ∧ *vl* = [] ∧ *vl1* = []

definition Δe :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δe *s vl s1 vl1* ≡
(∃ *cid*. *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *subPH*) ∧ *vl* ≠ []

lemma *istate-*$\Delta 1$:
assumes *B*: *B vl vl1*
shows $\Delta 1$ *istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: *unwind-cont* $\Delta 1$ {$\Delta 1$,$\Delta 2$,Δe}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 2$: *unwind-cont* $\Delta 2$ {$\Delta 2$,$\Delta 3$,Δe}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 3$: *unwind-cont* $\Delta 3$ {$\Delta 3$,Δe}
⟨*proof*⟩

definition *K1exit* **where**
K1exit s ≡ ∃ *cid*. *phase s cid* > *subPH* ∧ *PID* ∈∈ *paperIDs s cid*

lemma *invarNT-K1exit*: *invarNT K1exit*
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal K1exit v*
⟨*proof*⟩

lemma *unwind-exit-*Δe: *unwind-exit* Δe
⟨*proof*⟩

theorem *secure*: *secure*

⟨*proof*⟩

end
theory *Paper-All*
imports
Paper-Aut-PC
Paper-Aut
begin

end
theory *Review-Intro*
imports ../*Safety-Properties*
begin

6 Review Confidentiality

In this section, we prove confidentiality properties for the reviews of papers submitted to a conference. The secrets (values) of interest are therefore the different versions of a given review for a given paper, identified as the N'th review of the paper with id PID.

Here, we have three points of compromise between the bound and the trigger (which yield three properties). Let

- T1 denote "review authorship"
- T2 denote "PC membership having no conflict with that paper and the conference having moved to the discussion phase"
- T3 denote "PC membership or authorship and the conference having moved to the notification phase"

The three bound-trigger combinations are:

- weak trigger (T1 or T2 or T3) paired with strong bound (allowing to learn almost nothing)
- medium trigger (T1 or T2) paired with medium bound (allowing to learn the last edited version before notification)
- strong trigger (T1) paired with weak bound (allowing to learn the last edited version before discussion and all the later versions)

end

theory *Review-Value-Setup*
imports *Review-Intro*
begin

consts *PID* :: *paperID* **consts** *N* :: *nat*

(*PID*, *N*) identifies uniquely the review under scrutiny

6.1 Preliminaries

declare *updates-commute-paper*[*simp*]

Auxiliary definitions:

definition *eqExcNth* **where**
eqExcNth xs ys n $\equiv$
length xs = *length ys* $\wedge$ ($\forall$ *i* < *length xs*. *i* $\neq$ *n* $\longrightarrow$ *xs*!*i* = *ys*!*i*)

lemma *eqExcNth-eq*[*simp*,*intro*!]: *eqExcNth xs xs n*
⟨*proof*⟩

lemma *eqExcNth-sym*:
assumes *eqExcNth xs xs1 n*
shows *eqExcNth xs1 xs n*
⟨*proof*⟩

lemma *eqExcNth-trans*:
assumes *eqExcNth xs xs1 n* **and** *eqExcNth xs1 xs2 n*
shows *eqExcNth xs xs2 n*
⟨*proof*⟩

fun *eqExcD* :: *paper* $\Rightarrow$ *paper* $\Rightarrow$ *bool* **where**
eqExcD (*Paper name info ct reviews dis decs*)
(*Paper name1 info1 ct1 reviews1 dis1 decs1*) =
(*name* = *name1* $\wedge$ *info* = *info1* $\wedge$ *ct* = *ct1* $\wedge$ *dis* = *dis1* $\wedge$ *decs* = *decs1* $\wedge$
eqExcNth reviews reviews1 N)

lemma *eqExcD*:
eqExcD pap pap1 =
(*titlePaper pap* = *titlePaper pap1* $\wedge$ *abstractPaper pap* = *abstractPaper pap1* $\wedge$
contentPaper pap = *contentPaper pap1* $\wedge$
disPaper pap = *disPaper pap1* $\wedge$ *decsPaper pap* = *decsPaper pap1* $\wedge$
eqExcNth (*reviewsPaper pap*) (*reviewsPaper pap1*) *N*)
⟨*proof*⟩

lemma *eqExcD-eq*[*simp*,*intro*!]: *eqExcD pap pap*
⟨*proof*⟩

lemma *eqExcD-sym*:
assumes *eqExcD pap pap1*
shows *eqExcD pap1 pap*
⟨*proof*⟩

lemma *eqExcD-trans*:
assumes *eqExcD pap pap1* **and** *eqExcD pap1 pap2*
shows *eqExcD pap pap2*
⟨*proof*⟩

definition *eeqExcPID-N* **where**
eeqExcPID-N paps paps1 ≡
∀ *pid. if pid = PID then eqExcD (paps pid) (paps1 pid) else paps pid = paps1 pid*

lemma *eeqExcPID-N-eeq*[*simp*,*intro*!]: *eeqExcPID-N s s*
⟨*proof*⟩

lemma *eeqExcPID-N-sym*:
assumes *eeqExcPID-N s s1* **shows** *eeqExcPID-N s1 s*
⟨*proof*⟩

lemma *eeqExcPID-N-trans*:
assumes *eeqExcPID-N s s1* **and** *eeqExcPID-N s1 s2* **shows** *eeqExcPID-N s s2*
⟨*proof*⟩

lemma *eeqExcPID-N-imp*:
eeqExcPID-N paps paps1 ⟹ *eqExcD (paps PID) (paps1 PID)*
⟦*eeqExcPID-N paps paps1*; *pid* ≠ *PID*⟧ ⟹ *paps pid = paps1 pid*
⟨*proof*⟩

lemma *eeqExcPID-N-cong*:
assumes *eeqExcPID-N paps paps1*
and *pid = PID* ⟹ *eqExcD uu uu1*
and *pid* ≠ *PID* ⟹ *uu = uu1*
shows *eeqExcPID-N (paps (pid := uu)) (paps1(pid := uu1))*
⟨*proof*⟩

lemma *eeqExcPID-N-RDD*:
eeqExcPID-N paps paps1 ⟹
titlePaper (paps PID) = titlePaper (paps1 PID) ∧
abstractPaper (paps PID) = abstractPaper (paps1 PID) ∧
contentPaper (paps PID) = contentPaper (paps1 PID) ∧
disPaper (paps PID) = disPaper (paps1 PID) ∧
decsPaper (paps PID) = decsPaper (paps1 PID)
⟨*proof*⟩

The notion of two states being equal everywhere except on the the review $(N,\ PID)$:

definition *eqExcPID-N* :: *state* ⇒ *state* ⇒ *bool* **where**
eqExcPID-N s s1 ≡
confIDs s = confIDs s1 ∧ *conf s = conf s1* ∧
userIDs s = userIDs s1 ∧ *pass s = pass s1* ∧ *user s = user s1* ∧ *roles s = roles s1* ∧

paperIDs s = paperIDs s1
$\wedge$
eeqExcPID-N (paper s) (paper s1)
$\wedge$
pref s = pref s1 $\wedge$
voronkov s = voronkov s1 $\wedge$
news s = news s1 $\wedge$ *phase s = phase s1*

lemma *eqExcPID-N-eq*[*simp*,*intro*!]: *eqExcPID-N s s*
⟨*proof*⟩

lemma *eqExcPID-N-sym*:
assumes *eqExcPID-N s s1* **shows** *eqExcPID-N s1 s*
⟨*proof*⟩

lemma *eqExcPID-N-trans*:
assumes *eqExcPID-N s s1* **and** *eqExcPID-N s1 s2* **shows** *eqExcPID-N s s2*
⟨*proof*⟩

Implications from *eqExcPID-N*, including w.r.t. auxiliary operations:

lemma *eqExcPID-N-imp*:
eqExcPID-N s s1 $\Longrightarrow$
confIDs s = confIDs s1 $\wedge$ *conf s = conf s1* $\wedge$
userIDs s = userIDs s1 $\wedge$ *pass s = pass s1* $\wedge$ *user s = user s1* $\wedge$ *roles s = roles s1* $\wedge$
paperIDs s = paperIDs s1
$\wedge$
eeqExcPID-N (paper s) (paper s1)
$\wedge$
pref s = pref s1 $\wedge$
voronkov s = voronkov s1 $\wedge$
news s = news s1 $\wedge$ *phase s = phase s1* $\wedge$

getAllPaperIDs s = getAllPaperIDs s1 $\wedge$
isRev s cid uid pid = isRev s1 cid uid pid $\wedge$
getReviewIndex s cid uid pid = getReviewIndex s1 cid uid pid $\wedge$
getRevRole s cid uid pid = getRevRole s1 cid uid pid $\wedge$
length (reviewsPaper (paper s pid)) = length (reviewsPaper (paper s1 pid))
⟨*proof*⟩

lemma *eqExcPID-N-imp1*:
eqExcPID-N s s1 $\Longrightarrow$ *eqExcD (paper s pid) (paper s1 pid)*
eqExcPID-N s s1 $\Longrightarrow$ *pid* $\neq$ *PID* $\vee$ *PID* $\neq$ *pid* $\Longrightarrow$
    paper s pid = paper s1 pid $\wedge$
    getNthReview s pid n = getNthReview s1 pid n
⟨*proof*⟩

lemma *eqExcPID-N-imp2*:
assumes *eqExcPID-N s s1* **and** *pid* $\neq$ *PID* $\vee$ *PID* $\neq$ *pid*

shows *getReviewersReviews s cid pid = getReviewersReviews s1 cid pid*
⟨*proof*⟩

lemma *eqExcPID-N-imp3*:
eqExcPID-N s s1 ⟹ *pid* ≠ *PID* ∨ *PID* ≠ *pid* ∨ (*n* < *length* (*reviewsPaper* (*paper s PID*)) ∧ *n* ≠ *N*)
⟹
getNthReview s pid n = getNthReview s1 pid n
⟨*proof*⟩

lemma *eqExcPID-N-imp3′*:
assumes *s*: *reach s*
and *eqExcPID-N s s1* **and** *pid* ≠ *PID* ∨ (*isRevNth s cid uid pid n* ∧ *n* ≠ *N*)
shows *getNthReview s pid n = getNthReview s1 pid n*
⟨*proof*⟩

lemma *eqExcPID-N-RDD*:
eqExcPID-N s s1 ⟹
titlePaper (*paper s PID*) = *titlePaper* (*paper s1 PID*) ∧
abstractPaper (*paper s PID*) = *abstractPaper* (*paper s1 PID*) ∧
contentPaper (*paper s PID*) = *contentPaper* (*paper s1 PID*) ∧
disPaper (*paper s PID*) = *disPaper* (*paper s1 PID*) ∧
decsPaper (*paper s PID*) = *decsPaper* (*paper s1 PID*)
⟨*proof*⟩

lemma *eqExcPID-N-cong*[*simp*, *intro*]:
⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*confIDs* := *uu1*|)) (*s1* (|*confIDs* := *uu2*|))
⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*conf* := *uu1*|)) (*s1* (|*conf* := *uu2*|))

⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*userIDs* := *uu1*|)) (*s1* (|*userIDs* := *uu2*|))
⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*pass* := *uu1*|)) (*s1* (|*pass* := *uu2*|))
⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*user* := *uu1*|)) (*s1* (|*user* := *uu2*|))
⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*roles* := *uu1*|)) (*s1* (|*roles* := *uu2*|))

⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*paperIDs* := *uu1*|)) (*s1* (|*paperIDs* := *uu2*|))
⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *eeqExcPID-N uu1 uu2* ⟹ *eqExcPID-N* (*s* (|*paper* := *uu1*|)) (*s1* (|*paper* := *uu2*|))

⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*pref* := *uu1*|)) (*s1* (|*pref* := *uu2*|))
⋀ *uu1 uu2*. *eqExcPID-N s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID-N* (*s* (|*voronkov* :=

$uu1$ ⦈) ($s1$ ⦇$voronkov$:= $uu2$⦈)
$\bigwedge$ $uu1$ $uu2$. $eqExcPID\text{-}N$ s $s1$ $\Longrightarrow$ $uu1 = uu2$ $\Longrightarrow$ $eqExcPID\text{-}N$ (s ⦇$news$:= $uu1$⦈) ($s1$ ⦇$news$:= $uu2$⦈)
$\bigwedge$ $uu1$ $uu2$. $eqExcPID\text{-}N$ s $s1$ $\Longrightarrow$ $uu1 = uu2$ $\Longrightarrow$ $eqExcPID\text{-}N$ (s ⦇$phase$:= $uu1$⦈) ($s1$ ⦇$phase$:= $uu2$⦈)

⟨*proof*⟩

lemma *eqExcPID-N-Paper*:
assumes *s's1'*: *eqExcPID-N s s1*
and *paper s pid* = *Paper title abstract content reviews dis decs*
and *paper s1 pid* = *Paper title1 abstract1 content1 reviews1 dis1 decs1*
shows *title* = *title1* ∧ *abstract* = *abstract1* ∧ *content* = *content1* ∧ *dis* = *dis1* ∧ *decs* = *decs1*
⟨*proof*⟩

Auxiliary definitions for a slightly weaker equivalence relation defined below:

definition *eqExcNth2* **where**
eqExcNth2 rl rl1 n ≡
length rl = *length rl1* ∧
(∀ *i* < *length rl*. *i* ≠ *n* ⟶ *rl*!*i* = *rl1*!*i*) ∧
hd (*rl*!*n*) = *hd* (*rl1*!*n*)

lemma *eqExcNth2-eq*[*simp*,*intro*!]: *eqExcNth2 rl rl n*
⟨*proof*⟩

lemma *eqExcNth2-sym*:
assumes *eqExcNth2 rl rl1 n*
shows *eqExcNth2 rl1 rl n*
⟨*proof*⟩

lemma *eqExcNth2-trans*:
assumes *eqExcNth2 rl rl1 n* **and** *eqExcNth2 rl1 rl2 n*
shows *eqExcNth2 rl rl2 n*
⟨*proof*⟩

fun *eqExcD2* :: *paper* ⇒ *paper* ⇒ *bool* **where**
eqExcD2 (*Paper title abstract ct reviews dis decs*)
(*Paper title1 abstract1 ct1 reviews1 dis1 decs1*) =
(*title* = *title1* ∧ *abstract* = *abstract1* ∧ *ct* = *ct1* ∧ *dis* = *dis1* ∧ *decs* = *decs1* ∧
eqExcNth2 reviews reviews1 N)

lemma *eqExcD2*:
eqExcD2 pap pap1 =
(*titlePaper pap* = *titlePaper pap1* ∧ *abstractPaper pap* = *abstractPaper pap1* ∧
contentPaper pap = *contentPaper pap1* ∧
disPaper pap = *disPaper pap1* ∧ *decsPaper pap* = *decsPaper pap1* ∧
eqExcNth2 (*reviewsPaper pap*) (*reviewsPaper pap1*) *N*)
⟨*proof*⟩

lemma *eqExcD2-eq*[*simp*,*intro*!]: *eqExcD2 pap pap*
⟨*proof*⟩

lemma *eqExcD2-sym*:
assumes *eqExcD2 pap pap1*
shows *eqExcD2 pap1 pap*
⟨*proof*⟩

lemma *eqExcD2-trans*:
assumes *eqExcD2 pap pap1* **and** *eqExcD2 pap1 pap2*
shows *eqExcD2 pap pap2*
⟨*proof*⟩

definition *eeqExcPID-N2* **where**
eeqExcPID-N2 paps paps1 ≡
∀ *pid. if pid* = *PID then eqExcD2* (*paps pid*) (*paps1 pid*) *else paps pid* = *paps1 pid*

lemma *eeqExcPID-N2-eeq*[*simp*,*intro*!]: *eeqExcPID-N2 s s*
⟨*proof*⟩

lemma *eeqExcPID-N2-sym*:
assumes *eeqExcPID-N2 s s1* **shows** *eeqExcPID-N2 s1 s*
⟨*proof*⟩

lemma *eeqExcPID-N2-trans*:
assumes *eeqExcPID-N2 s s1* **and** *eeqExcPID-N2 s1 s2* **shows** *eeqExcPID-N2 s s2*
⟨*proof*⟩

lemma *eeqExcPID-N2-imp*:
eeqExcPID-N2 paps paps1 ⟹ *eqExcD2* (*paps PID*) (*paps1 PID*)
⟦*eeqExcPID-N2 paps paps1*; *pid* ≠ *PID*⟧ ⟹ *paps pid* = *paps1 pid*
⟨*proof*⟩

lemma *eeqExcPID-N2-cong*:
assumes *eeqExcPID-N2 paps paps1*
and *pid* = *PID* ⟹ *eqExcD2 uu uu1*
and *pid* ≠ *PID* ⟹ *uu* = *uu1*
shows *eeqExcPID-N2* (*paps* (*pid* := *uu*)) (*paps1*(*pid* := *uu1*))
⟨*proof*⟩

lemma *eeqExcPID-N2-RDD*:
eeqExcPID-N2 paps paps1 ⟹
titlePaper (*paps PID*) = *titlePaper* (*paps1 PID*) ∧
abstractPaper (*paps PID*) = *abstractPaper* (*paps1 PID*) ∧
contentPaper (*paps PID*) = *contentPaper* (*paps1 PID*) ∧
disPaper (*paps PID*) = *disPaper* (*paps1 PID*) ∧
decsPaper (*paps PID*) = *decsPaper* (*paps1 PID*)

⟨*proof*⟩

A weaker state equivalence that allows differences in old versions of the score and comments of the review (N, PID). It is used for the confidentiality property that does not cover PC members in the discussion phase, when they will learn about scores and comments.

definition *eqExcPID-N2* :: *state* ⇒ *state* ⇒ *bool* **where**
eqExcPID-N2 s s1 ≡
confIDs s = *confIDs s1* ∧ *conf s* = *conf s1* ∧
userIDs s = *userIDs s1* ∧ *pass s* = *pass s1* ∧ *user s* = *user s1* ∧ *roles s* = *roles s1* ∧
paperIDs s = *paperIDs s1*
∧
eeqExcPID-N2 (*paper s*) (*paper s1*)
∧
pref s = *pref s1* ∧
voronkov s = *voronkov s1* ∧
news s = *news s1* ∧ *phase s* = *phase s1*

lemma *eqExcPID-N2-eq*[*simp*,*intro*!]: *eqExcPID-N2 s s*
⟨*proof*⟩

lemma *eqExcPID-N2-sym*:
assumes *eqExcPID-N2 s s1* **shows** *eqExcPID-N2 s1 s*
⟨*proof*⟩

lemma *eqExcPID-N2-trans*:
assumes *eqExcPID-N2 s s1* **and** *eqExcPID-N2 s1 s2* **shows** *eqExcPID-N2 s s2*
⟨*proof*⟩

Implications from *eqExcPID-N2*, including w.r.t. auxiliary operations:

lemma *eqExcPID-N2-imp*:
eqExcPID-N2 s s1 ⟹
confIDs s = *confIDs s1* ∧ *conf s* = *conf s1* ∧
userIDs s = *userIDs s1* ∧ *pass s* = *pass s1* ∧ *user s* = *user s1* ∧ *roles s* = *roles s1* ∧
paperIDs s = *paperIDs s1*
∧
eeqExcPID-N2 (*paper s*) (*paper s1*)
∧
pref s = *pref s1* ∧
voronkov s = *voronkov s1* ∧
news s = *news s1* ∧ *phase s* = *phase s1* ∧

getAllPaperIDs s = *getAllPaperIDs s1* ∧
isRev s cid uid pid = *isRev s1 cid uid pid* ∧
getReviewIndex s cid uid pid = *getReviewIndex s1 cid uid pid* ∧
getRevRole s cid uid pid = *getRevRole s1 cid uid pid* ∧
length (*reviewsPaper* (*paper s pid*)) = *length* (*reviewsPaper* (*paper s1 pid*))

$\langle proof \rangle$

lemma *eqExcPID-N2-imp1*:
eqExcPID-N2 s s1 $\Longrightarrow$ *eqExcD2* (*paper s pid*) (*paper s1 pid*)
eqExcPID-N2 s s1 $\Longrightarrow$ *pid* $\neq$ *PID* $\vee$ *PID* $\neq$ *pid* $\Longrightarrow$
paper s pid = *paper s1 pid* $\wedge$
getNthReview s pid n = *getNthReview s1 pid n*
$\langle proof \rangle$

lemma *eqExcPID-N2-imp2*:
assumes *eqExcPID-N2 s s1* **and** *pid* $\neq$ *PID* $\vee$ *PID* $\neq$ *pid*
shows *getReviewersReviews s cid pid* = *getReviewersReviews s1 cid pid*
$\langle proof \rangle$

lemma *eqExcPID-N2-eqExcPID-N*:
eqExcPID-N2 s s1 $\Longrightarrow$ *eqExcPID-N s s1*
$\langle proof \rangle$

lemma *eqExcPID-N2-imp3*:
eqExcPID-N2 s s1 $\Longrightarrow$ *pid* $\neq$ *PID* $\vee$ *PID* $\neq$ *pid* $\vee$ (*n* < *length* (*reviewsPaper* (*paper s PID*)) $\wedge$ *n* $\neq$ *N*)
$\Longrightarrow$
getNthReview s pid n = *getNthReview s1 pid n*
$\langle proof \rangle$

lemma *eqExcPID-N2-imp3'*:
assumes *s*: *reach s*
and *eqExcPID-N2 s s1* **and** *pid* $\neq$ *PID* $\vee$ (*isRevNth s cid uid pid n* $\wedge$ *n* $\neq$ *N*)
shows *getNthReview s pid n* = *getNthReview s1 pid n*
$\langle proof \rangle$

lemma *eqExcPID-N2-imp33*:
assumes *eqExcPID-N2 s s1*
shows *hd* (*getNthReview s pid N*) = *hd* (*getNthReview s1 pid N*)
$\langle proof \rangle$

lemma *eqExcPID-N2-RDD*:
eqExcPID-N2 s s1 $\Longrightarrow$
titlePaper (*paper s PID*) = *titlePaper* (*paper s1 PID*) $\wedge$
abstractPaper (*paper s PID*) = *abstractPaper* (*paper s1 PID*) $\wedge$
contentPaper (*paper s PID*) = *contentPaper* (*paper s1 PID*) $\wedge$
disPaper (*paper s PID*) = *disPaper* (*paper s1 PID*) $\wedge$
decsPaper (*paper s PID*) = *decsPaper* (*paper s1 PID*)
$\langle proof \rangle$

lemma *eqExcPID-N2-cong*[*simp*, *intro*]:
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (|*confIDs* := *uu1*|)) (*s1* (|*confIDs* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*conf* := *uu1*\|)) (*s1* (\|*conf* := *uu2*\|))

$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*userIDs* := *uu1*\|)) (*s1* (\|*userIDs* := *uu2*\|))
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*pass* := *uu1*\|)) (*s1* (\|*pass* := *uu2*\|))
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*user* := *uu1*\|)) (*s1* (\|*user* := *uu2*\|))
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*roles* := *uu1*\|)) (*s1* (\|*roles* := *uu2*\|))

$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*paperIDs* := *uu1*\|)) (*s1* (\|*paperIDs* := *uu2*\|))
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *eeqExcPID-N2 uu1 uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*paper* := *uu1*\|)) (*s1* (\|*paper* := *uu2*\|))

$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*pref* := *uu1*\|)) (*s1* (\|*pref* := *uu2*\|))
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*voronkov* := *uu1*\|)) (*s1* (\|*voronkov* := *uu2*\|))
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*news* := *uu1*\|)) (*s1* (\|*news* := *uu2*\|))
$\bigwedge$ *uu1 uu2*. *eqExcPID-N2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID-N2* (*s* (\|*phase* := *uu1*\|)) (*s1* (\|*phase* := *uu2*\|))

⟨*proof*⟩

lemma *eqExcPID-N2-Paper*:
assumes *s's1'*: *eqExcPID-N2 s s1*
and *paper s pid* = *Paper title abstract content reviews dis decs*
and *paper s1 pid* = *Paper title1 abstract1 content1 reviews1 dis1 decs1*
shows *title* = *title1* ∧ *abstract* = *abstract1* ∧ *content* = *content1* ∧ *dis* = *dis1* ∧ *decs* = *decs1*
⟨*proof*⟩

lemma *eqExcPID-N2-step*:
assumes *ss1*: *eqExcPID-N2 s s1*
and *step*: *step s a* = (*ou*,*s'*)
and *step1*: *step s1 a* = (*ou1*,*s1'*)
and *s*: *reach s* **and** *r*: *isRevNth s cid uid PID N*
shows *eqExcPID-N2 s' s1'*
⟨*proof*⟩

6.2 Value Setup

fun φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans* - (*Uact* (*uReview cid uid p pid n rc*)) *ou* -) =

$(pid = PID \wedge n = N \wedge ou = outOK)$
|
φ (*Trans* - (*UUact* (*uuReview cid uid p pid n rc*)) *ou* -) =
$(pid = PID \wedge n = N \wedge ou = outOK)$
|
φ - = *False*

lemma φ-*def2*:
φ (*Trans s a ou s'*) =
(*ou* = *outOK* $\wedge$
($\exists$ *cid uid p rc*.
a = *Uact* (*uReview cid uid p PID N rc*)
$\vee$
a = *UUact* (*uuReview cid uid p PID N rc*)
))
⟨*proof*⟩

lemma *uReview-uuReview-step-eqExcPID-N*:
assumes *a*:
a = *Uact* (*uReview cid uid p PID N rc*) $\vee$
a = *UUact* (*uuReview cid uid p PID N rc*)
and *step s a* = (*ou*,*s'*)
shows *eqExcPID-N s s'*
⟨*proof*⟩

lemma φ-*step-eqExcPID-N*:
assumes φ: φ (*Trans s a ou s'*)
and *s*: *step s a* = (*ou*,*s'*)
shows *eqExcPID-N s s'*
⟨*proof*⟩

lemma *eqExcPID-N-step*:
assumes *s's1'*: *eqExcPID-N s s1*
and *step*: *step s a* = (*ou*,*s'*)
and *step1*: *step s1 a* = (*ou1*,*s1'*)
shows *eqExcPID-N s' s1'*
⟨*proof*⟩

lemma *eqExcPID-N-step-*φ*-imp*:
assumes *ss1*: *eqExcPID-N s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and φ: φ (*Trans s a ou s'*)
shows φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqExcPID-N-step-*φ:
assumes *s's1'*: *eqExcPID-N s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
shows φ (*Trans s a ou s'*) = φ (*Trans s1 a ou1 s1'*)

⟨*proof*⟩

lemma *eqExcPID-N2-step-φ-imp*:
assumes *ss1*: *eqExcPID-N2 s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *r*: *N* < *length* (*reviewsPaper* (*paper s PID*))
and φ: φ (*Trans s a ou s'*)
shows φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqExcPID-N2-step-φ*:
assumes *s*: *reach s* **and** *s1*: *reach s1*
and *ss1*: *eqExcPID-N2 s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
shows φ (*Trans s a ou s'*) = φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

end
theory *Review-RAut*
imports ../*Observation-Setup Review-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*
begin

6.3 Confidentiality protection from users who are not the review's author

We verify the following property:

A group UIDs of users learn nothing about the various updates of the N'th review of a paper PID except for the last edited version before discussion and all the later versions unless a user in UIDs is that review's author.

type-synonym *value* = *phase* ∗ *rcontent*

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *value* **where**
f (*Trans s* (*Uact* (*uReview cid uid p pid n rc*)) - -) = (*phase s cid*, *rc*)
|
f (*Trans s* (*UUact* (*uuReview cid uid p pid n rc*)) - -) = (*phase s cid*, *rc*)

fun *T* :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
T (*Trans* - - *ou s'*) =
(∃ *uid* ∈ *UIDs*. *isREVNth s' uid PID N*)

declare *T.simps* [*simp del*]

definition *B* :: *value list* ⇒ *value list* ⇒ *bool* **where**

$B\ vl\ vl1 \equiv$
$\exists\ ul\ ul1\ wl.$
$vl = (map\ (Pair\ revPH)\ ul)\ @\ (map\ (Pair\ disPH)\ wl) \wedge$
$vl1 = (map\ (Pair\ revPH)\ ul1)\ @\ (map\ (Pair\ disPH)\ wl) \wedge$
$ul \neq [] \wedge ul1 \neq [] \wedge last\ ul = last\ ul1$

interpretation *BD-Security-IO* **where**
istate = *istate* **and** *step* = *step* **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
⟨*proof*⟩

lemma *reachNT-non-isRevNth*:
assumes *reachNT s* **and** *uid* ∈ *UIDs*
shows ¬ *isRevNth s cid uid PID N*
⟨*proof*⟩

lemma *P-φ-γ*:
assumes *1*: *reachNT s* **and** *2*: *step s a* = (*ou*,*s'*) φ (*Trans s a ou s'*)
shows ¬ γ (*Trans s a ou s'*)
⟨*proof*⟩

lemma *eqExcPID-N-step-out*:
assumes *s's1'*: *eqExcPID-N s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *sT*: *reachNT s* **and** *s1*: *reach s1*
and *PID*: *PID* ∈∈ *paperIDs s cid*
and *ph*: *phase s cid* = *revPH*
and *UIDs*: *userOfA a* ∈ *UIDs*
shows *ou* = *ou1*
⟨*proof*⟩

lemma *eeqExcPID-N-imp-eq*:
assumes *eeqExcPID-N paps paps1*
and *reviewsPaper* (*paps PID*) ! *N* = *reviewsPaper* (*paps1 PID*) ! *N*
shows *paps* = *paps1*
⟨*proof*⟩

lemma *eqExcPID-N-imp-eq*:
assumes *e*: *eqExcPID-N s s1*
and *reviewsPaper* (*paper s PID*) ! *N* = *reviewsPaper* (*paper s1 PID*) ! *N*
shows *s* = *s1*
⟨*proof*⟩

lemma *eqExcPID-N-step-eq*:
assumes *s*: *reach s* **and** *ss1*: *eqExcPID-N s s1*
and *a*: *a* = *Uact* (*uReview cid uid p PID N rc*)
and *step*: *step s a* = (*outOK*, *s'*) **and** *step1*: *step s1 a* = (*ou'*, *s1'*)

shows $s' = s1'$
⟨*proof*⟩

definition $\Delta 1$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 1$ *s vl s1 vl1* ≡
(∀ *cid*. *PID* ∈∈ *paperIDs s cid* ⟶ *phase s cid* < *revPH*) ∧
s = *s1* ∧ *B vl vl1*

definition $\Delta 2$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 2$ *s vl s1 vl1* ≡
∃ *cid*.
PID ∈∈ *paperIDs s cid* ∧ *phase s cid* = *revPH* ∧ ¬ (∃ *uid*. *isREVNth s uid PID N*) ∧
s = *s1* ∧ *B vl vl1*

definition $\Delta 3$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 3$ *s vl s1 vl1* ≡
∃ *cid uid*.
PID ∈∈ *paperIDs s cid* ∧ *phase s cid* = *revPH* ∧ *isREVNth s uid PID N* ∧
eqExcPID-N s s1 ∧ *B vl vl1*

definition $\Delta 4$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 4$ *s vl s1 vl1* ≡
∃ *cid uid*.
PID ∈∈ *paperIDs s cid* ∧ *phase s cid* ≥ *revPH* ∧ *isREVNth s uid PID N* ∧
s = *s1* ∧ (∃ *wl*. *vl* = *map* (*Pair disPH*) *wl* ∧ *vl1* = *map* (*Pair disPH*) *wl*)

definition Δe :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δe *s vl s1 vl1* ≡
vl ≠ [] ∧
(
(∃ *cid*. *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH* ∧ ¬ (∃ *uid*. *isREVNth s uid PID N*))
∨
(∃ *cid*. *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH* ∧ *fst* (*hd vl*) = *revPH*)
)

lemma *istate*-$\Delta 1$:
assumes *B*: *B vl vl1*
shows $\Delta 1$ *istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont*-$\Delta 1$: *unwind-cont* $\Delta 1$ {$\Delta 1$,$\Delta 2$,Δe}
⟨*proof*⟩

lemma *unwind-cont*-$\Delta 2$: *unwind-cont* $\Delta 2$ {$\Delta 2$,$\Delta 3$,Δe}
⟨*proof*⟩

lemma *unwind-cont*-$\Delta 3$: *unwind-cont* $\Delta 3$ {$\Delta 3$,$\Delta 4$,Δe}

⟨proof⟩

lemma *unwind-cont-*$\Delta 4$: *unwind-cont* $\Delta 4$ $\{\Delta 4, \Delta e\}$
⟨proof⟩

definition *K2exit* **where**
K2exit cid s ≡
PID ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH* ∧ ¬ (∃ *uid*. *isRevNth s cid uid PID N*)

lemma *invarNT-K2exit*: *invarNT* (*K2exit cid*)
⟨proof⟩

lemma *noVal-K2exit*: *noVal* (*K2exit cid*) *v*
⟨proof⟩

definition *K3exit* **where**
K3exit cid s ≡ *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH*

lemma *invarNT-K3exit*: *invarNT* (*K3exit cid*)
⟨proof⟩

lemma *noVal-K3exit*: *noVal* (*K3exit cid*) (*revPH*,*u*)
⟨proof⟩

lemma *unwind-exit-*Δe: *unwind-exit* Δe
⟨proof⟩

theorem *secure*: *secure*
⟨proof⟩

end
theory *Review-RAut-NCPC*
imports *../Observation-Setup Review-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*
begin

6.4 Confidentiality protection from users who are not the review's author or a PC member

We verify the following property:

A group of users UIDs learn nothing about the various updates of the N'th review of a paper PID except for the last edited version before notification unless/until one of the following holds:

- a user in UIDs is the review's author, or

- a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper, and the conference moves to the discussion phase.

type-synonym *value* = *rcontent*

fun f :: $(state,act,out)$ *trans* $\Rightarrow$ *value* **where**
f (*Trans* - (*Uact* (*uReview cid uid p pid n rc*)) - -) = *rc*
|
f (*Trans* - (*UUact* (*uuReview cid uid p pid n rc*)) - -) = *rc*

fun T :: $(state,act,out)$ *trans* $\Rightarrow$ *bool* **where**
T (*Trans* - - *ou* s') =
($\exists$ *uid* $\in$ *UIDs*.
isREVNth s' *uid PID N*
$\vee$
($\exists$ *cid*. *PID* $\in\in$ *paperIDs* s' *cid* $\wedge$ *isPC* s' *cid uid* $\wedge$ *pref* s' *uid PID* $\neq$ *Conflict* $\wedge$ *phase* s' *cid* $\geq$ *disPH*)
)

declare *T.simps* [*simp del*]

definition B :: *value list* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
B vl vl1 $\equiv$ *vl* $\neq$ [] $\wedge$ *vl1* $\neq$ [] $\wedge$ *last vl* = *last vl1*

interpretation *BD-Security-IO* **where**
istate = *istate* **and** *step* = *step* **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma *reachNT-non-isRevNth-isPC-isChair*:
assumes *reachNT s* **and** *uid* $\in$ *UIDs*
shows
$\neg$ *isRevNth s cid uid PID N* $\wedge$
(*PID* $\in\in$ *paperIDs s cid* $\wedge$ *isPC s cid uid* $\longrightarrow$ *pref s uid PID* = *Conflict* $\vee$ *phase s cid* < *disPH*) $\wedge$
(*PID* $\in\in$ *paperIDs s cid* $\wedge$ *isChair s cid uid* $\longrightarrow$ *pref s uid PID* = *Conflict* $\vee$ *phase s cid* < *disPH*)
$\langle proof \rangle$

lemma *T-*φ*-*γ:
assumes *1*: *reachNT s* **and** *2*: *step s a* = (*ou*,s') φ (*Trans s a ou* s')
shows $\neg$ γ (*Trans s a ou* s')
$\langle proof \rangle$

lemma *eqExcPID-N-step-out*:
assumes $s's1'$: *eqExcPID-N s s1*
and *step*: *step s a* = (*ou*,s') **and** *step1*: *step s1 a* = (*ou1*,$s1'$)
and *sP*: *reachNT s* **and** *s1*: *reach s1*

and PID: $PID \in\in paperIDs\ s\ cid$
and ph: $phase\ s\ cid = revPH \lor phase\ s\ cid = disPH$
and $UIDs$: $userOfA\ a \in UIDs$
shows $ou = ou1$
$\langle proof \rangle$

lemma $eqExcPID\text{-}N2\text{-}step\text{-}out$:
assumes $ss1$: $eqExcPID\text{-}N2\ s\ s1$
and $step$: $step\ s\ a = (ou,s')$ **and** $step1$: $step\ s1\ a = (ou1,s1')$
and sP: $reachNT\ s$ **and** $s1$: $reach\ s1$
and r: $isRevNth\ s\ cid\ uid\ PID\ N$
and ph: $phase\ s\ cid \geq revPH$
and $UIDs$: $userOfA\ a \in UIDs$
and $decs\text{-}exit$: $(reviewsPaper\ (paper\ s\ PID))!N \neq [] \land (reviewsPaper\ (paper\ s1\ PID))!N \neq []$
shows $ou = ou1$
$\langle proof \rangle$

lemma $eqExcPID\text{-}N\text{-}step\text{-}eqExcPID\text{-}N2$:
assumes rs: $reach\ s$
and a: $a = Uact\ (uReview\ cid\ uid\ p\ PID\ N\ rc) \lor$
$a = UUact\ (uuReview\ cid\ uid\ p\ PID\ N\ rc)$ (**is** $?L \lor ?R$)
and $ss1$: $eqExcPID\text{-}N\ s\ s1$
and $step$: $step\ s\ a = (outOK,s')$ **and** $step1$: $step\ s1\ a = (outOK,s1')$
shows $eqExcPID\text{-}N2\ s'\ s1'$
$\langle proof \rangle$

lemma $eqExcPID\text{-}N\text{-}step\text{-}\varphi\text{-}eqExcPID\text{-}N2$:
assumes rs: $reach\ s$
and $ss1$: $eqExcPID\text{-}N\ s\ s1$
and $step$: $step\ s\ a = (ou,s')$ **and** $step1$: $step\ s1\ a = (ou1,s1')$
and φ: $\varphi\ (Trans\ s\ a\ ou\ s')$
shows $eqExcPID\text{-}N2\ s'\ s1'$
$\langle proof \rangle$

definition $\Delta 1$ :: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 1\ s\ vl\ s1\ vl1 \equiv$
$(\forall\ cid.\ PID \in\in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < revPH) \land$
$s = s1 \land B\ vl\ vl1$

definition $\Delta 2$ :: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 2\ s\ vl\ s1\ vl1 \equiv$
$\exists\ cid.$
$PID \in\in paperIDs\ s\ cid \land phase\ s\ cid = revPH \land \neg\ (\exists\ uid.\ isREVNth\ s\ uid\ PID\ N) \land$
$s = s1 \land B\ vl\ vl1$

definition $\Delta 3$ :: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 3\ s\ vl\ s1\ vl1 \equiv$

$\exists$ *cid uid*.
PID $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\in$ $\{$*revPH*, *disPH*$\}$ $\wedge$ *isREVNth s uid PID N* $\wedge$
eqExcPID-N s s1 $\wedge$ *B vl vl1*

definition $\Delta 4$ :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$\Delta 4$ *s vl s1 vl1* $\equiv$
$\exists$ *cid uid*.
PID $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *revPH* $\wedge$ *isREVNth s uid PID N* $\wedge$
(*reviewsPaper* (*paper s PID*))!*N* $\neq$ [] $\wedge$ (*reviewsPaper* (*paper s1 PID*))!*N* $\neq$ [] $\wedge$
eqExcPID-N2 s s1 $\wedge$ *vl* = [] $\wedge$ *vl1* = []

definition Δe :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δe *s vl s1 vl1* $\equiv$
vl $\neq$ [] $\wedge$
(
($\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* > *revPH* $\wedge$ $\neg$ ($\exists$ *uid*. *isREVNth s uid PID N*))
$\vee$
($\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* > *disPH*)
)

lemma *istate-*$\Delta 1$:
assumes *B*: *B vl vl1*
shows $\Delta 1$ *istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: *unwind-cont* $\Delta 1$ $\{\Delta 1, \Delta 2, \Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 2$: *unwind-cont* $\Delta 2$ $\{\Delta 2, \Delta 3, \Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 3$: *unwind-cont* $\Delta 3$ $\{\Delta 3, \Delta 4, \Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 4$: *unwind-cont* $\Delta 4$ $\{\Delta 4, \Delta e\}$
⟨*proof*⟩

definition *K1exit* **where**
K1exit cid s $\equiv$ *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* > *revPH* $\wedge$ $\neg$ ($\exists$ *uid*. *isRevNth s cid uid PID N*)

lemma *invarNT-K1exit*: *invarNT* (*K1exit cid*)
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
⟨*proof*⟩

definition *K2exit* **where**
K2exit cid s ≡ *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *disPH*

lemma *invarNT-K2exit*: *invarNT* (*K2exit cid*)
⟨*proof*⟩

lemma *noVal-K2exit*: *noVal* (*K2exit cid*) *v*
⟨*proof*⟩

lemma *unwind-exit-Δe*: *unwind-exit* Δ*e*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Review-RAut-NCPC-PAut*
imports ../*Observation-Setup Review-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*
begin

6.5 Confidentiality from users who are not the review's author, a PC member, or an author of the paper

We verify the following property:

A group of users UIDs learn nothing about the various updates to the N'th review of a paper PID (save for the inexistence of any updates) unless/until

- a user in UIDs is the review's author, or
- a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the discussion phase, or
- a user in UIDs become a PC member in the paper's conference or an author of the paper and the conference moves to the notification phase

type-synonym *value* = *rcontent*

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *value* **where**
f (*Trans* - (*Uact* (*uReview cid uid p pid n rc*)) - -) = *rc*
|
f (*Trans* - (*UUact* (*uuReview cid uid p pid n rc*)) - -) = *rc*

fun T :: $(state,act,out)$ $trans \Rightarrow bool$ **where**
T $(Trans$ - - ou $s')$ =
$(\exists$ $uid \in UIDs.$
$isREVNth$ s' uid PID N
$\vee$
$(\exists$ $cid.$ $PID \in\in paperIDs$ s' $cid \wedge isPC$ s' cid $uid \wedge pref$ s' uid $PID \neq Conflict$
$\wedge$ $phase$ s' $cid \geq disPH)$
$\vee$
$(\exists$ $cid.$ $PID \in\in paperIDs$ s' $cid \wedge isPC$ s' cid $uid \wedge phase$ s' $cid \geq notifPH)$
$\vee$
$isAUT$ s' uid $PID \wedge (\exists$ $cid.$ $PID \in\in paperIDs$ s' $cid \wedge phase$ s' $cid \geq notifPH)$
)

declare $T.simps$ $[simp$ $del]$

definition B :: $value$ $list \Rightarrow value$ $list \Rightarrow bool$ **where**
B vl $vl1$ $\equiv$ $vl \neq []$

interpretation *BD-Security-IO* **where**
$istate = istate$ **and** $step = step$ **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma *reachNT-non-isPC-isChair*:
assumes $reachNT$ s **and** $uid \in UIDs$
shows
$\neg$ $isRevNth$ s cid uid PID N $\wedge$
$(PID \in\in paperIDs$ s $cid \wedge isPC$ s cid $uid \longrightarrow$
$(pref$ s uid $PID = Conflict \vee phase$ s $cid < disPH) \wedge phase$ s $cid < notifPH)$
$\wedge$
$(PID \in\in paperIDs$ s $cid \wedge isChair$ s cid $uid \longrightarrow$
$(pref$ s uid $PID = Conflict \vee phase$ s $cid < disPH) \wedge phase$ s $cid < notifPH)$
$\wedge$
$(isAut$ s cid uid $PID \longrightarrow phase$ s $cid < notifPH)$
$\langle proof \rangle$

lemma *T-φ-γ*:
assumes *1*: $reachNT$ s **and** *2*: $step$ s $a = (ou,s')$ φ $(Trans$ s a ou $s')$
shows $\neg$ γ $(Trans$ s a ou $s')$
$\langle proof \rangle$

lemma *eqExcPID-N-step-out*:
assumes $s's1'$: $eqExcPID\text{-}N$ s $s1$
and *step*: $step$ s $a = (ou,s')$ **and** *step1*: $step$ $s1$ $a = (ou1,s1')$
and *sT*: $reachNT$ s **and** *s1*: $reach$ $s1$
and *PID*: $PID \in\in paperIDs$ s cid
and *UIDs*: $userOfA$ $a \in UIDs$
shows $ou = ou1$
$\langle proof \rangle$

definition $\Delta 1$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 1$ *s vl s1 vl1* ≡
(∀ *cid*. *PID* ∈∈ *paperIDs s cid* ⟶ *phase s cid* < *revPH*) ∧ *s* = *s1* ∧ *B vl vl1*

definition $\Delta 2$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 2$ *s vl s1 vl1* ≡
∃ *cid*.
PID ∈∈ *paperIDs s cid* ∧ *phase s cid* = *revPH* ∧
¬ (∃ *uid*. *isREVNth s uid PID N*) ∧
s = *s1* ∧ *B vl vl1*

definition $\Delta 3$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 3$ *s vl s1 vl1* ≡
∃ *cid uid*. *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* = *revPH* ∧ *isREVNth s uid PID N* ∧ *eqExcPID-N s s1*

definition $\Delta 4$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 4$ *s vl s1 vl1* ≡
∃ *cid*. *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH* ∧ *eqExcPID-N s s1* ∧ *vl1* = []

definition Δe :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δe *s vl s1 vl1* ≡
vl ≠ [] ∧
(∃ *cid*. *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH* ∧ ¬ (∃ *uid*. *isREVNth s uid PID N*))

lemma *istate-*$\Delta 1$:
assumes *B*: *B vl vl1*
shows $\Delta 1$ *istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: *unwind-cont* $\Delta 1$ {$\Delta 1$,$\Delta 2$,Δe}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 2$: *unwind-cont* $\Delta 2$ {$\Delta 2$,$\Delta 3$,Δe}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 3$: *unwind-cont* $\Delta 3$ {$\Delta 3$,$\Delta 4$,Δe}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 4$: *unwind-cont* $\Delta 4$ {$\Delta 4$,Δe}
⟨*proof*⟩

definition *K1exit* **where**

K1exit cid s ≡ *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH* ∧ ¬ (∃ *uid*. *isRevNth s cid uid PID N*)

lemma *invarNT-K1exit*: *invarNT* (*K1exit cid*)
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
⟨*proof*⟩

lemma *unwind-exit-*Δ*e*: *unwind-exit* Δ*e*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Review-All*
imports
Review-RAut
Review-RAut-NCPC
Review-RAut-NCPC-PAut
begin

end
theory *Discussion-Intro*
imports ../*Safety-Properties*
begin

7 Discussion Confidentiality

In this section, we prove confidentiality for the discussion log (with comments made by PC members) on submitted papers. The secrets (values) of interest are therefore the different updates of (i.e., comments posted as part of) the discussion of a given paper with id PID.

Here, we have only one point of compromise between the bound and the trigger (which yields one property): the trigger being "PC membership having no conflict with that paper and the conference having moved to the discussion stage" and the bound allowing to learn almost nothing.

end
theory *Discussion-Value-Setup*
imports *Discussion-Intro*
begin

The ID of the paper under scrutiny:

consts *PID* :: *paperID*

7.1 Preliminaries

declare *updates-commute-paper*[*simp*]

fun *eqExcD* :: *paper* ⇒ *paper* ⇒ *bool* **where**
eqExcD (*Paper title abstract ct reviews dis decs*)
  (*Paper title1 abstract1 ct1 reviews1 dis1 decs1*) =
(*title* = *title1* ∧ *abstract* = *abstract1* ∧ *ct* = *ct1* ∧ *reviews* = *reviews1* ∧ *decs* = *decs1*)

lemma *eqExcD*:
eqExcD pap pap1 =
(*titlePaper pap* = *titlePaper pap1* ∧ *abstractPaper pap* = *abstractPaper pap1* ∧
contentPaper pap = *contentPaper pap1* ∧
reviewsPaper pap = *reviewsPaper pap1* ∧ *decsPaper pap* = *decsPaper pap1*)
⟨*proof*⟩

lemma *eqExcD-eq*[*simp*,*intro*!]: *eqExcD pap pap*
⟨*proof*⟩

lemma *eqExcD-sym*:
assumes *eqExcD pap pap1*
shows *eqExcD pap1 pap*
⟨*proof*⟩

lemma *eqExcD-trans*:
assumes *eqExcD pap pap1* **and** *eqExcD pap1 pap2*
shows *eqExcD pap pap2*
⟨*proof*⟩

definition *eeqExcPID* **where**
eeqExcPID paps paps1 ≡
∀ *pid*. *if pid* = *PID then eqExcD* (*paps pid*) (*paps1 pid*) *else paps pid* = *paps1 pid*

lemma *eeqExcPID-eeq*[*simp*,*intro*!]: *eeqExcPID s s*
⟨*proof*⟩

lemma *eeqExcPID-sym*:
assumes *eeqExcPID s s1* **shows** *eeqExcPID s1 s*
⟨*proof*⟩

lemma *eeqExcPID-trans*:
assumes *eeqExcPID s s1* **and** *eeqExcPID s1 s2* **shows** *eeqExcPID s s2*
⟨*proof*⟩

lemma *eeqExcPID-imp*:
eeqExcPID paps paps1 ⟹ *eqExcD* (*paps PID*) (*paps1 PID*)
⟦*eeqExcPID paps paps1*; *pid* ≠ *PID*⟧ ⟹ *paps pid* = *paps1 pid*

$\langle proof \rangle$

lemma *eeqExcPID-cong*:
assumes *eeqExcPID paps paps1*
and $pid = PID \Longrightarrow eqExcD\ uu\ uu1$
and $pid \neq PID \Longrightarrow uu = uu1$
shows $eeqExcPID\ (paps\ (pid := uu))\ (paps1(pid := uu1))$
$\langle proof \rangle$

lemma *eeqExcPID-RDD*:
$eeqExcPID\ paps\ paps1 \Longrightarrow$
$titlePaper\ (paps\ PID) = titlePaper\ (paps1\ PID) \wedge$
$abstractPaper\ (paps\ PID) = abstractPaper\ (paps1\ PID) \wedge$
$contentPaper\ (paps\ PID) = contentPaper\ (paps1\ PID) \wedge$
$reviewsPaper\ (paps\ PID) = reviewsPaper\ (paps1\ PID) \wedge$
$decsPaper\ (paps\ PID) = decsPaper\ (paps1\ PID)$
$\langle proof \rangle$

definition $eqExcPID :: state \Rightarrow state \Rightarrow bool$ **where**
$eqExcPID\ s\ s1 \equiv$
$confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$
$userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1 \wedge roles\ s = roles$
$s1 \wedge$
$paperIDs\ s = paperIDs\ s1$
$\wedge$
$eeqExcPID\ (paper\ s)\ (paper\ s1)$
$\wedge$
$pref\ s = pref\ s1 \wedge$
$voronkov\ s = voronkov\ s1 \wedge$
$news\ s = news\ s1 \wedge phase\ s = phase\ s1$

lemma *eqExcPID-eq*[*simp*,*intro*!]: *eqExcPID s s*
$\langle proof \rangle$

lemma *eqExcPID-sym*:
assumes *eqExcPID s s1* **shows** *eqExcPID s1 s*
$\langle proof \rangle$

lemma *eqExcPID-trans*:
assumes *eqExcPID s s1* **and** *eqExcPID s1 s2* **shows** *eqExcPID s s2*
$\langle proof \rangle$

lemma *eqExcPID-imp*:
$eqExcPID\ s\ s1 \Longrightarrow$
$confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$
$userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1 \wedge roles\ s = roles$
$s1 \wedge$

$paperIDs\ s = paperIDs\ s1$
$\wedge$
$eeqExcPID\ (paper\ s)\ (paper\ s1)$
$\wedge$
$pref\ s = pref\ s1\ \wedge$
$voronkov\ s = voronkov\ s1\ \wedge$
$news\ s = news\ s1\ \wedge\ phase\ s = phase\ s1\ \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1\ \wedge$
$isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid\ \wedge$
$getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid\ \wedge$
$getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
$\langle proof \rangle$

lemma *eqExcPID-imp1*:
$eqExcPID\ s\ s1 \Longrightarrow eqExcD\ (paper\ s\ pid)\ (paper\ s1\ pid)$
$eqExcPID\ s\ s1 \Longrightarrow pid \neq PID \vee PID \neq pid \Longrightarrow$
$\quad paper\ s\ pid = paper\ s1\ pid\ \wedge$
$\quad getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
$\langle proof \rangle$

lemma *eqExcPID-imp2*:
assumes $eqExcPID\ s\ s1$ **and** $pid \neq PID \vee PID \neq pid$
shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
$\langle proof \rangle$

lemma *eqExcPID-RDD*:
$eqExcPID\ s\ s1 \Longrightarrow$
$titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID)\ \wedge$
$abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID)\ \wedge$
$contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID)\ \wedge$
$reviewsPaper\ (paper\ s\ PID) = reviewsPaper\ (paper\ s1\ PID)\ \wedge$
$decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$
$\langle proof \rangle$

lemma *eqExcPID-cong*[*simp*, *intro*]:
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (|confIDs := uu1|))\ (s1\ (|confIDs := uu2|))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (|conf := uu1|))\ (s1\ (|conf := uu2|))$

$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (|userIDs := uu1|))\ (s1\ (|userIDs := uu2|))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (|pass := uu1|))\ (s1\ (|pass := uu2|))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (|user := uu1|))\ (s1\ (|user := uu2|))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (|roles := uu1|))\ (s1\ (|roles := uu2|))$

$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *uu1* $=$ *uu2* $\Longrightarrow$ *eqExcPID* (*s* (\|*paperIDs* := *uu1*\|)) (*s1* (\|*paperIDs* := *uu2*\|))
$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *eeqExcPID uu1 uu2* $\Longrightarrow$ *eqExcPID* (*s* (\|*paper* := *uu1*\|)) (*s1* (\|*paper* := *uu2*\|))

$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *uu1* $=$ *uu2* $\Longrightarrow$ *eqExcPID* (*s* (\|*pref* := *uu1*\|)) (*s1* (\|*pref* := *uu2*\|))
$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *uu1* $=$ *uu2* $\Longrightarrow$ *eqExcPID* (*s* (\|*voronkov* := *uu1*\|)) (*s1* (\|*voronkov* := *uu2*\|))
$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *uu1* $=$ *uu2* $\Longrightarrow$ *eqExcPID* (*s* (\|*news* := *uu1*\|)) (*s1* (\|*news* := *uu2*\|))
$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *uu1* $=$ *uu2* $\Longrightarrow$ *eqExcPID* (*s* (\|*phase* := *uu1*\|)) (*s1* (\|*phase* := *uu2*\|))
⟨*proof*⟩

lemma *eqExcPID-Paper*:
assumes *s′s1′*: *eqExcPID s s1*
and *paper s pid* = *Paper title abstract content reviews dis decs*
and *paper s1 pid* = *Paper title1 abstract1 content1 reviews1 dis1 decs1*
shows *title* = *title1* ∧ *abstract* = *abstract1* ∧ *content* = *content1* ∧ *reviews* = *reviews1* ∧ *decs* = *decs1*
⟨*proof*⟩

7.2 Value Setup

type-synonym *value* = *string*

fun φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans* - (*UUact* (*uuDis cid uid p pid com*)) *ou* -) = (*pid* = *PID* ∧ *ou* = *outOK*)
|
φ - = *False*

lemma φ*-def2*:
φ (*Trans s a ou s′*) = (∃ *cid uid p com. a* = *UUact* (*uuDis cid uid p PID com*) ∧ *ou* = *outOK*)
⟨*proof*⟩

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *value* **where**
f (*Trans* - (*UUact* (*uuDis cid uid p pid com*)) - -) = *com*

lemma *UUact-uuDis-step-eqExcPID*:
assumes *a*: *a* = *UUact* (*uuDis cid uid p PID com*)
and *step s a* = (*ou*,*s′*)
shows *eqExcPID s s′*
⟨*proof*⟩

lemma φ*-step-eqExcPID*:
assumes φ: φ (*Trans s a ou s′*)

and *s*: *step s a* = (*ou*,*s'*)
shows *eqExcPID s s'*
⟨*proof*⟩

lemma *eqExcPID-step*:
assumes *s's1'*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*)
and *step1*: *step s1 a* = (*ou1*,*s1'*)
shows *eqExcPID s' s1'*
⟨*proof*⟩

lemma *eqExcPID-step-φ-imp*:
assumes *s's1'*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *φ*: *φ* (*Trans s a ou s'*)
shows *φ* (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqExcPID-step-φ*:
assumes *s's1'*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
shows *φ* (*Trans s a ou s'*) = *φ* (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

end
theory *Discussion-NCPC*
imports ../*Observation-Setup Discussion-Value-Setup Bounded-Deducibility-Security.Compositional-Reasonin*
begin

7.3 Confidentiality protection from non-PC-members

We verify the following property:

A group of users UIDs learn nothing about the various updates of a paper's discussion (i.e., about the comments being posted on a paper by the PC members) (save for the non-existence of any edit) unless/until a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the discussion phase.

fun *T* :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
T (*Trans - - ou s'*) =
(∃ *uid* ∈ *UIDs*. ∃ *cid*.
PID ∈∈ *paperIDs s' cid* ∧ *isPC s' cid uid* ∧ *pref s' uid PID* ≠ *Conflict* ∧ *phase s' cid* ≥ *disPH*
)

declare $T.simps$ [$simp\ del$]

definition $B :: value\ list \Rightarrow value\ list \Rightarrow bool$ **where**
$B\ vl\ vl1 \equiv vl \neq []$

interpretation $BD\text{-}Security\text{-}IO$ **where**
$istate = istate$ **and** $step = step$ **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma $reachNT\text{-}non\text{-}isPC\text{-}isChair$:
assumes $reachNT\ s$ **and** $uid \in UIDs$
shows
$(PID \in\in paperIDs\ s\ cid \wedge isPC\ s\ cid\ uid \wedge phase\ s\ cid \geq disPH \longrightarrow pref\ s\ uid\ PID = Conflict) \wedge$
$(PID \in\in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \wedge phase\ s\ cid \geq disPH \longrightarrow pref\ s\ uid\ PID = Conflict)$
$\langle proof \rangle$

lemma $T\text{-}\varphi\text{-}\gamma$:
assumes 1: $reachNT\ s$ **and** 2: $step\ s\ a = (ou,s')\ \varphi\ (Trans\ s\ a\ ou\ s')$
shows $\neg\ \gamma\ (Trans\ s\ a\ ou\ s')$
$\langle proof \rangle$

lemma $eqExcPID\text{-}step\text{-}out$:
assumes $s's1'$: $eqExcPID\ s\ s1$
and $step$: $step\ s\ a = (ou,s')$ **and** $step1$: $step\ s1\ a = (ou1,s1')$
and sT: $reachNT\ s$ **and** $s1$: $reach\ s1$
and PID: $PID \in\in paperIDs\ s\ cid$
and $UIDs$: $userOfA\ a \in UIDs$
shows $ou = ou1$
$\langle proof \rangle$

definition $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 1\ s\ vl\ s1\ vl1 \equiv$
$(\forall\ cid.\ PID \in\in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < disPH) \wedge s = s1 \wedge B\ vl\ vl1$

definition $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 2\ s\ vl\ s1\ vl1 \equiv$
$\exists\ cid\ uid.$
$PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$
$isPC\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict$
$\wedge\ eqExcPID\ s\ s1$

definition $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 3\ s\ vl\ s1\ vl1 \equiv$

$\exists$ *cid. PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $>$ *disPH* $\wedge$ *eqExcPID s s1* $\wedge$ *vl1* $=$ []

definition Δe :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δe *s vl s1 vl1* $\equiv$
vl $\neq$ [] $\wedge$
($\exists$ *cid. PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *disPH* $\wedge$
$\neg$ ($\exists$ *uid. isPC s cid uid* $\wedge$ *pref s uid PID* $\neq$ *Conflict*)
)

lemma *init-*$\Delta 1$:
assumes *B*: *B vl vl1*
shows $\Delta 1$ *istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: *unwind-cont* $\Delta 1$ $\{\Delta 1,\Delta 2,\Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 2$: *unwind-cont* $\Delta 2$ $\{\Delta 2,\Delta 3,\Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 3$: *unwind-cont* $\Delta 3$ $\{\Delta 3,\Delta e\}$
⟨*proof*⟩

definition *K1exit* **where**
K1exit cid s $\equiv$
(*PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *disPH* $\wedge$
$\neg$ ($\exists$ *uid. isPC s cid uid* $\wedge$ *pref s uid PID* $\neq$ *Conflict*))

lemma *invarNT-K1exit*: *invarNT* (*K1exit cid*)
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
⟨*proof*⟩

lemma *unwind-exit-*Δe: *unwind-exit* Δe
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Discussion-All*
imports
Discussion-NCPC
begin

end
theory *Decision-Intro*
imports *../Safety-Properties*
begin

8 Decision Confidentiality

In this section, we prove confidentiality properties for the accept-reject decision of papers submitted to a conference. The secrets (values) of interest are therefore the different updates of the decision of a given paper with id PID.

Here, we have two points of compromise between the bound and the trigger (which yield two properties). Let

- T1 denote "PC membership having no conflict with that paper and the conference having moved to the discussion stage"
- T2 denote "PC membership or authorship, and the conference having moved to the notification phase"

The two trigger-bound combinations are:

- weak trigger (T1 or T2) paired with strong bound (allowing to learn almost nothing)
- strong trigger (T1) paired with weak bound (allowing to learn the last updated version of the decision)

end
theory *Decision-Value-Setup*
imports *Decision-Intro*
begin

The ID of the paper under scrutiny:

consts *PID* :: *paperID*

8.1 Preliminaries

declare *updates-commute-paper*[*simp*]

fun *eqExcD* :: *paper* ⇒ *paper* ⇒ *bool* **where**
eqExcD (*Paper title abstract ct reviews dis decs*)
(*Paper title1 abstract1 ct1 reviews1 dis1 decs1*) =
(*title* = *title1* ∧ *abstract* = *abstract1* ∧ *ct* = *ct1* ∧ *reviews* = *reviews1* ∧ *dis* = *dis1*)

lemma *eqExcD*:
eqExcD pap pap1 =
(*titlePaper pap* = *titlePaper pap1* ∧ *abstractPaper pap* = *abstractPaper pap1* ∧
contentPaper pap = *contentPaper pap1* ∧
reviewsPaper pap = *reviewsPaper pap1* ∧ *disPaper pap* = *disPaper pap1*)
⟨*proof*⟩

lemma *eqExcD-eq*[*simp*,*intro*!]: *eqExcD pap pap*
⟨*proof*⟩

lemma *eqExcD-sym*:
assumes *eqExcD pap pap1*
shows *eqExcD pap1 pap*
⟨*proof*⟩

lemma *eqExcD-trans*:
assumes *eqExcD pap pap1* **and** *eqExcD pap1 pap2*
shows *eqExcD pap pap2*
⟨*proof*⟩

definition *eeqExcPID* **where**
eeqExcPID paps paps1 ≡
∀ *pid*. *if pid* = *PID then eqExcD* (*paps pid*) (*paps1 pid*) *else paps pid* = *paps1 pid*

lemma *eeqExcPID-eeq*[*simp*,*intro*!]: *eeqExcPID s s*
⟨*proof*⟩

lemma *eeqExcPID-sym*:
assumes *eeqExcPID s s1* **shows** *eeqExcPID s1 s*
⟨*proof*⟩

lemma *eeqExcPID-trans*:
assumes *eeqExcPID s s1* **and** *eeqExcPID s1 s2* **shows** *eeqExcPID s s2*
⟨*proof*⟩

lemma *eeqExcPID-imp*:
eeqExcPID paps paps1 ⟹ *eqExcD* (*paps PID*) (*paps1 PID*)
⟦*eeqExcPID paps paps1*; *pid* ≠ *PID*⟧ ⟹ *paps pid* = *paps1 pid*
⟨*proof*⟩

lemma *eeqExcPID-cong*:
assumes *eeqExcPID paps paps1*
and *pid* = *PID* ⟹ *eqExcD uu uu1*
and *pid* ≠ *PID* ⟹ *uu* = *uu1*
shows *eeqExcPID* (*paps* (*pid* := *uu*)) (*paps1*(*pid* := *uu1*))
⟨*proof*⟩

lemma *eeqExcPID-RDD*:

eeqExcPID paps paps1 $\Longrightarrow$
titlePaper (*paps PID*) = *titlePaper* (*paps1 PID*) $\wedge$
abstractPaper (*paps PID*) = *abstractPaper* (*paps1 PID*) $\wedge$
contentPaper (*paps PID*) = *contentPaper* (*paps1 PID*) $\wedge$
reviewsPaper (*paps PID*) = *reviewsPaper* (*paps1 PID*) $\wedge$
disPaper (*paps PID*) = *disPaper* (*paps1 PID*)
⟨*proof*⟩

definition *eqExcPID* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *bool* **where**
eqExcPID s s1 $\equiv$
confIDs s = *confIDs s1* $\wedge$ *conf s* = *conf s1* $\wedge$
userIDs s = *userIDs s1* $\wedge$ *pass s* = *pass s1* $\wedge$ *user s* = *user s1* $\wedge$ *roles s* = *roles s1* $\wedge$
paperIDs s = *paperIDs s1*
$\wedge$
eeqExcPID (*paper s*) (*paper s1*)
$\wedge$
pref s = *pref s1* $\wedge$
voronkov s = *voronkov s1* $\wedge$
news s = *news s1* $\wedge$ *phase s* = *phase s1*

lemma *eqExcPID-eq*[*simp*,*intro*!]: *eqExcPID s s*
⟨*proof*⟩

lemma *eqExcPID-sym*:
assumes *eqExcPID s s1* **shows** *eqExcPID s1 s*
⟨*proof*⟩

lemma *eqExcPID-trans*:
assumes *eqExcPID s s1* **and** *eqExcPID s1 s2* **shows** *eqExcPID s s2*
⟨*proof*⟩

lemma *eqExcPID-imp*:
eqExcPID s s1 $\Longrightarrow$
confIDs s = *confIDs s1* $\wedge$ *conf s* = *conf s1* $\wedge$
userIDs s = *userIDs s1* $\wedge$ *pass s* = *pass s1* $\wedge$ *user s* = *user s1* $\wedge$ *roles s* = *roles s1* $\wedge$
paperIDs s = *paperIDs s1*
$\wedge$
eeqExcPID (*paper s*) (*paper s1*)
$\wedge$
pref s = *pref s1* $\wedge$
voronkov s = *voronkov s1* $\wedge$
news s = *news s1* $\wedge$ *phase s* = *phase s1* $\wedge$

getAllPaperIDs s = *getAllPaperIDs s1* $\wedge$
isRev s cid uid pid = *isRev s1 cid uid pid* $\wedge$

$getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
$getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
$\langle proof \rangle$

lemma *eqExcPID-imp1*:
$eqExcPID\ s\ s1 \Longrightarrow eqExcD\ (paper\ s\ pid)\ (paper\ s1\ pid)$
$eqExcPID\ s\ s1 \Longrightarrow pid \neq PID \vee PID \neq pid \Longrightarrow$
$paper\ s\ pid = paper\ s1\ pid \wedge$
$getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
$\langle proof \rangle$

lemma *eqExcPID-imp2*:
assumes $eqExcPID\ s\ s1$ **and** $pid \neq PID \vee PID \neq pid$
shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
$\langle proof \rangle$

lemma *eqExcPID-RDD*:
$eqExcPID\ s\ s1 \Longrightarrow$
$titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
$abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
$contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$
$reviewsPaper\ (paper\ s\ PID) = reviewsPaper\ (paper\ s1\ PID) \wedge$
$disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID)$
$\langle proof \rangle$

lemma *eqExcPID-cong*[*simp*, *intro*]:
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|confIDs := uu1|\!))\ (s1\ (\!|confIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|conf := uu1|\!))\ (s1\ (\!|conf := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|userIDs := uu1|\!))\ (s1\ (\!|userIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|pass := uu1|\!))\ (s1\ (\!|pass := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|user := uu1|\!))\ (s1\ (\!|user := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|roles := uu1|\!))\ (s1\ (\!|roles := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|paperIDs := uu1|\!))\ (s1\ (\!|paperIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow eeqExcPID\ uu1\ uu2 \Longrightarrow eqExcPID\ (s\ (\!|paper := uu1|\!))\ (s1\ (\!|paper := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|pref := uu1|\!))\ (s1\ (\!|pref := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqExcPID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqExcPID\ (s\ (\!|voronkov := uu1|\!))\ (s1\ (\!|voronkov := uu2|\!))$

$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *uu1* $=$ *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*news* := *uu1*|)) (*s1* (|*news* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqExcPID s s1* $\Longrightarrow$ *uu1* $=$ *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*phase* := *uu1*|)) (*s1* (|*phase* := *uu2*|))
⟨*proof*⟩

lemma *eqExcPID-Paper*:
assumes *s's1'*: *eqExcPID s s1*
and *paper s pid* = *Paper title abstract content reviews dis decs*
and *paper s1 pid* = *Paper title1 abstract1 content1 reviews1 dis1 decs1*
shows *title* = *title1* ∧ *abstract* = *abstract1* ∧ *content* = *content1* ∧ *reviews* = *reviews1* ∧ *dis* = *dis1*
⟨*proof*⟩

Weaker equivalence relations that allow differences in the final decision. This is used for verifying the confidentiality property that only protects earlier updates to the decision.

fun *eqExcD2* :: *paper* ⇒ *paper* ⇒ *bool* **where**
eqExcD2 (*Paper title abstract ct reviews dis decs*)
(*Paper title1 abstract1 ct1 reviews1 dis1 decs1*) =
(*title* = *title1* ∧ *abstract* = *abstract1* ∧ *ct* = *ct1* ∧ *reviews* = *reviews1* ∧ *dis* = *dis1* ∧
hd decs = *hd decs1*)

lemma *eqExcD2*:
eqExcD2 pap pap1 =
(*titlePaper pap* = *titlePaper pap1* ∧ *abstractPaper pap* = *abstractPaper pap1* ∧
contentPaper pap = *contentPaper pap1* ∧
reviewsPaper pap = *reviewsPaper pap1* ∧ *disPaper pap* = *disPaper pap1* ∧
hd (*decsPaper pap*) = *hd* (*decsPaper pap1*)
)
⟨*proof*⟩

lemma *eqExcD2-eq*[*simp*,*intro*!]: *eqExcD2 pap pap*
⟨*proof*⟩

lemma *eqExcD2-sym*:
assumes *eqExcD2 pap pap1*
shows *eqExcD2 pap1 pap*
⟨*proof*⟩

lemma *eqExcD2-trans*:
assumes *eqExcD2 pap pap1* **and** *eqExcD2 pap1 pap2*
shows *eqExcD2 pap pap2*
⟨*proof*⟩

definition *eeqExcPID2* **where**
eeqExcPID2 paps paps1 ≡

$\forall$ *pid. if pid = PID then eqExcD2 (paps pid) (paps1 pid) else paps pid = paps1 pid*

lemma *eeqExcPID2-eeq[simp,intro!]: eeqExcPID2 s s*
⟨*proof*⟩

lemma *eeqExcPID2-sym*:
assumes *eeqExcPID2 s s1* **shows** *eeqExcPID2 s1 s*
⟨*proof*⟩

lemma *eeqExcPID2-trans*:
assumes *eeqExcPID2 s s1* **and** *eeqExcPID2 s1 s2* **shows** *eeqExcPID2 s s2*
⟨*proof*⟩

lemma *eeqExcPID2-imp*:
eeqExcPID2 paps paps1 $\Longrightarrow$ *eqExcD2 (paps PID) (paps1 PID)*
$[\![$*eeqExcPID2 paps paps1; pid* $\neq$ *PID*$]\!]$ $\Longrightarrow$ *paps pid = paps1 pid*
⟨*proof*⟩

lemma *eeqExcPID2-cong*:
assumes *eeqExcPID2 paps paps1*
and *pid = PID* $\Longrightarrow$ *eqExcD2 uu uu1*
and *pid* $\neq$ *PID* $\Longrightarrow$ *uu = uu1*
shows *eeqExcPID2 (paps (pid := uu)) (paps1(pid := uu1))*
⟨*proof*⟩

lemma *eeqExcPID2-RDD*:
eeqExcPID2 paps paps1 $\Longrightarrow$
titlePaper (paps PID) = titlePaper (paps1 PID) $\wedge$
abstractPaper (paps PID) = abstractPaper (paps1 PID) $\wedge$
contentPaper (paps PID) = contentPaper (paps1 PID) $\wedge$
reviewsPaper (paps PID) = reviewsPaper (paps1 PID) $\wedge$
disPaper (paps PID) = disPaper (paps1 PID) $\wedge$
hd (decsPaper (paps PID)) = hd (decsPaper (paps PID))
⟨*proof*⟩

definition *eqExcPID2* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *bool* **where**
eqExcPID2 s s1 $\equiv$
confIDs s = confIDs s1 $\wedge$ *conf s = conf s1* $\wedge$
userIDs s = userIDs s1 $\wedge$ *pass s = pass s1* $\wedge$ *user s = user s1* $\wedge$ *roles s = roles s1* $\wedge$
paperIDs s = paperIDs s1
$\wedge$
eeqExcPID2 (paper s) (paper s1)
$\wedge$
pref s = pref s1 $\wedge$
voronkov s = voronkov s1 $\wedge$
news s = news s1 $\wedge$ *phase s = phase s1*

lemma *eqExcPID2-eq*[*simp*,*intro*!]: *eqExcPID2 s s*
⟨*proof*⟩

lemma *eqExcPID2-sym*:
assumes *eqExcPID2 s s1* **shows** *eqExcPID2 s1 s*
⟨*proof*⟩

lemma *eqExcPID2-trans*:
assumes *eqExcPID2 s s1* **and** *eqExcPID2 s1 s2* **shows** *eqExcPID2 s s2*
⟨*proof*⟩

lemma *eqExcPID2-imp*:
$eqExcPID2\ s\ s1 \Longrightarrow$
$confIDs\ s = confIDs\ s1 \wedge conf\ s = conf\ s1 \wedge$
$userIDs\ s = userIDs\ s1 \wedge pass\ s = pass\ s1 \wedge user\ s = user\ s1 \wedge roles\ s = roles$
$s1 \wedge$
$paperIDs\ s = paperIDs\ s1$
$\wedge$
$eeqExcPID2\ (paper\ s)\ (paper\ s1)$
$\wedge$
$pref\ s = pref\ s1 \wedge$
$voronkov\ s = voronkov\ s1 \wedge$
$news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1 \wedge$
$isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$
$getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
$getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid$
⟨*proof*⟩

lemma *eqExcPID2-imp1*:
$eqExcPID2\ s\ s1 \Longrightarrow eqExcD2\ (paper\ s\ pid)\ (paper\ s1\ pid)$
$eqExcPID2\ s\ s1 \Longrightarrow pid \neq PID \vee PID \neq pid \Longrightarrow$
$paper\ s\ pid = paper\ s1\ pid \wedge$
$getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
⟨*proof*⟩

lemma *eqExcPID2-imp2*:
assumes $eqExcPID2\ s\ s1$ **and** $pid \neq PID \vee PID \neq pid$
shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
⟨*proof*⟩

lemma *eqExcPID2-RDD*:
$eqExcPID2\ s\ s1 \Longrightarrow$
$titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
$abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
$contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$

reviewsPaper (*paper s PID*) = *reviewsPaper* (*paper s1 PID*) ∧
disPaper (*paper s PID*) = *disPaper* (*paper s1 PID*)
⟨*proof*⟩

lemma *eqExcPID2-cong*[*simp*, *intro*]:
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*confIDs* := *uu1*|)) (*s1* (|*confIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*conf* := *uu1*|)) (*s1* (|*conf* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*userIDs* := *uu1*|)) (*s1* (|*userIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*pass* := *uu1*|)) (*s1* (|*pass* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*user* := *uu1*|)) (*s1* (|*user* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*roles* := *uu1*|)) (*s1* (|*roles* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*paperIDs* := *uu1*|)) (*s1* (|*paperIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *eeqExcPID2 uu1 uu2* ⟹ *eqExcPID2* (*s* (|*paper* := *uu1*|)) (*s1* (|*paper* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*pref* := *uu1*|)) (*s1* (|*pref* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*voronkov* := *uu1*|)) (*s1* (|*voronkov* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*news* := *uu1*|)) (*s1* (|*news* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* ⟹ *uu1* = *uu2* ⟹ *eqExcPID2* (*s* (|*phase* := *uu1*|)) (*s1* (|*phase* := *uu2*|))
⟨*proof*⟩

lemma *eqExcPID2-Paper*:
assumes *s's1'*: *eqExcPID2 s s1*
and *paper s pid* = *Paper title abstract content reviews dis decs*
and *paper s1 pid* = *Paper title1 abstract1 content1 reviews1 dis1 decs1*
shows *title* = *title1* ∧ *abstract* = *abstract1* ∧ *content* = *content1* ∧ *reviews* = *reviews1* ∧
dis = *dis1*
⟨*proof*⟩

8.2 Value Setup

type-synonym *value* = *decision*

fun φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans* - (*UUact* (*uuDec cid uid p pid dec*)) *ou* -) = (*pid* = *PID* ∧ *ou* = *outOK*)

|
φ - = *False*

lemma *φ-def2*:
φ (*Trans s a ou s'*) = ($\exists$ *cid uid p dec. a* = *UUact* (*uuDec cid uid p PID dec*) $\wedge$ *ou* = *outOK*)
⟨*proof*⟩

fun *f* :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *value* **where**
f (*Trans* - (*UUact* (*uuDec cid uid p pid dec*)) - -) = *dec*

lemma *UUact-uuDec-step-eqExcPID*:
assumes *a*: *a* = *UUact* (*uuDec cid uid p PID dec*)
and *step s a* = (*ou*,*s'*)
shows *eqExcPID s s'*
⟨*proof*⟩

lemma *φ-step-eqExcPID*:
assumes φ: φ (*Trans s a ou s'*)
and *s*: *step s a* = (*ou*,*s'*)
shows *eqExcPID s s'*
⟨*proof*⟩

lemma *eqExcPID-step*:
assumes *s's1'*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*)
and *step1*: *step s1 a* = (*ou1*,*s1'*)
shows *eqExcPID s' s1'*
⟨*proof*⟩

lemma *eqExcPID-step-φ-imp*:
assumes *s's1'*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and φ: φ (*Trans s a ou s'*)
shows φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqExcPID-step-φ*:
assumes *s's1'*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
shows φ (*Trans s a ou s'*) = φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqExcPID2-step*:
assumes *s's1'*: *eqExcPID2 s s1*
and *step*: *step s a* = (*ou*,*s'*)

and *step1*: *step s1 a* = (*ou1*,*s1*′)
shows *eqExcPID2 s*′ *s1*′
⟨*proof*⟩

lemma *eqExcPID2-step-φ-imp*:
assumes *s*′*s1*′: *eqExcPID2 s s1*
and *step*: *step s a* = (*ou*,*s*′) **and** *step1*: *step s1 a* = (*ou1*,*s1*′)
and *φ*: *φ* (*Trans s a ou s*′)
shows *φ* (*Trans s1 a ou1 s1*′)
⟨*proof*⟩

lemma *eqExcPID2-step-φ*:
assumes *s*′*s1*′: *eqExcPID2 s s1*
and *step*: *step s a* = (*ou*,*s*′) **and** *step1*: *step s1 a* = (*ou1*,*s1*′)
shows *φ* (*Trans s a ou s*′) = *φ* (*Trans s1 a ou1 s1*′)
⟨*proof*⟩

end
theory *Decision-NCPC*
imports ../*Observation-Setup Decision-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*
begin

8.3 Confidentiality protection from non-PC-members

We verify the following property:

A group of users UIDs learn nothing about the various updates of a paper's decision except for the last edited version unless/until a user in UIDs becomes PC member in the paper's conference having no conflict with that paper and the conference moves to the decision stage.

fun *T* :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
T (*Trans* - - *ou s*′) =
(∃ *uid* ∈ *UIDs*. ∃ *cid*.
PID ∈∈ *paperIDs s*′ *cid* ∧ *isPC s*′ *cid uid* ∧ *pref s*′ *uid PID* ≠ *Conflict* ∧
phase s′ *cid* ≥ *disPH*
)

declare *T.simps* [*simp del*]

definition *B* :: *value list* ⇒ *value list* ⇒ *bool* **where**
B vl vl1 ≡ *vl* ≠ [] ∧ *vl1* ≠ [] ∧ *last vl* = *last vl1*

interpretation *BD-Security-IO* **where**
istate = *istate* **and** *step* = *step* **and**
φ = *φ* **and** *f* = *f* **and** *γ* = *γ* **and** *g* = *g* **and** *T* = *T* **and** *B* = *B*

⟨*proof*⟩

lemma *reachNT-non-isPC-isChair*:
assumes *reachNT s* **and** *uid* ∈ *UIDs*
shows
(*PID* ∈∈ *paperIDs s cid* ∧ *isPC s cid uid* ∧ *phase s cid* ≥ *disPH* ⟶ *pref s uid PID* = *Conflict*) ∧
(*PID* ∈∈ *paperIDs s cid* ∧ *isChair s cid uid* ∧ *phase s cid* ≥ *disPH* ⟶ *pref s uid PID* = *Conflict*)
⟨*proof*⟩

lemma *T-φ-γ*:
assumes *1*: *reachNT s* **and** *2*: *step s a* = (*ou*,*s'*) *φ* (*Trans s a ou s'*)
shows ¬ *γ* (*Trans s a ou s'*)
⟨*proof*⟩

lemma *eqExcPID2-eqExcPID*:
eqExcPID2 s s1 ⟹ *eqExcPID s s1*
⟨*proof*⟩

lemma *eqExcPID-step-out*:
assumes *s's1'*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *sT*: *reachNT s* **and** *s1*: *reach s1*
and *PID*: *PID* ∈∈ *paperIDs s cid*
and *ph*: *phase s cid* = *disPH*
and *UIDs*: *userOfA a* ∈ *UIDs*
shows *ou* = *ou1*
⟨*proof*⟩

lemma *eqExcPID2-step-out*:
assumes *ss1*: *eqExcPID2 s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *sT*: *reachNT s* **and** *s1*: *reach s1*
and *PID*: *PID* ∈∈ *paperIDs s cid*
and *ph*: *phase s cid* ≥ *disPH*
and *UIDs*: *userOfA a* ∈ *UIDs*
and *decs-exit*: *decsPaper* (*paper s PID*) ≠ [] *decsPaper* (*paper s1 PID*) ≠ []
shows *ou* = *ou1*
⟨*proof*⟩

lemma *eqExcPID-step-eqExcPID2*:
assumes *a*: *a* = *UUact* (*uuDec cid uid p PID dec*)
and *ss1*: *eqExcPID s s1*
and *step*: *step s a* = (*outOK*,*s'*) **and** *step1*: *step s1 a* = (*outOK*,*s1'*)
and *s*: *reach s reach s1* **and** *PID*: *PID* ∈∈ *paperIDs s cid* **and** *ph*: *phase s cid* < *notifPH*
shows *eqExcPID2 s' s1'*
⟨*proof*⟩

lemma *eqExcPID-step-*φ*-eqExcPID2*:
assumes *ss1*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and φ: φ (*Trans s a ou s'*)
and *s*: *reach s reach s1* **and** *PID*: *PID* $\in\in$ *paperIDs s cid* **and** *ph*: *phase s cid* $\leq$ *disPH*
shows *eqExcPID2 s' s1'*
⟨*proof*⟩

definition Δ*1* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*1 s vl s1 vl1* $\equiv$
($\forall$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\longrightarrow$ *phase s cid* < *disPH*) $\wedge$
s = *s1* $\wedge$ *B vl vl1*

definition Δ*2* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*2 s vl s1 vl1* $\equiv$
$\exists$ *cid uid*.
PID $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* = *disPH* $\wedge$
isChair s cid uid $\wedge$ *pref s uid PID* $\neq$ *Conflict* $\wedge$
eqExcPID s s1 $\wedge$ *B vl vl1*

definition Δ*3* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*3 s vl s1 vl1* $\equiv$
$\exists$ *cid*.
PID $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *disPH* $\wedge$
decsPaper (*paper s PID*) $\neq$ [] $\wedge$ *decsPaper* (*paper s1 PID*) $\neq$ [] $\wedge$ *eqExcPID2 s s1* $\wedge$
vl = [] $\wedge$ *vl1* = []

definition Δ*e* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*e s vl s1 vl1* $\equiv$
vl $\neq$ [] $\wedge$
(($\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* > *disPH*)
$\vee$
($\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *disPH* $\wedge$
$\neg$ ($\exists$ *uid*. *isChair s cid uid* $\wedge$ *pref s uid PID* $\neq$ *Conflict*))
)

lemma *istate-*Δ*1*:
assumes *B*: *B vl vl1*
shows Δ*1 istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-*Δ*1*: *unwind-cont* Δ*1* {Δ*1*,Δ*2*,Δ*e*}
⟨*proof*⟩

lemma *unwind-cont-*Δ*2*: *unwind-cont* Δ*2* {Δ*2*,Δ*3*,Δ*e*}
⟨*proof*⟩

lemma *unwind-cont-*Δ*3*: *unwind-cont* Δ*3* {Δ*3*,Δ*e*}
⟨*proof*⟩

definition *K1exit* **where**
K1exit cid s ≡
phase s cid ≥ *disPH* ∧ *PID* ∈∈ *paperIDs s cid* ∧ ¬ (∃ *uid*. *isChair s cid uid* ∧ *pref s uid PID* ≠ *Conflict*)

lemma *invarNT-K1exit*: *invarNT* (*K1exit cid*)
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
⟨*proof*⟩

definition *K2exit* **where**
K2exit cid s ≡ *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *disPH*

lemma *invarNT-K2exit*: *invarNT* (*K2exit cid*)
⟨*proof*⟩

lemma *noVal-K2exit*: *noVal* (*K2exit cid*) *v*
⟨*proof*⟩

lemma *unwind-exit-*Δ*e*: *unwind-exit* Δ*e*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Decision-NCPC-Aut*
imports ../*Observation-Setup Decision-Value-Setup Bounded-Deducibility-Security.Compositional-Reasoning*
begin

8.4 Confidentiality protection from users who are not PC members or authors of the paper

We verify the following property:

A group of users UIDs learn nothing about the various updates of a paper's decision (save for the non-existence of any update) unless/until one of the following happens:

- a user in UIDs becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the discussion phase, or

- a user in UIDs becomes a PC member in the paper's conference or an author of the paper, and the conference moves to the notification phase

fun T :: $(state,act,out)$ $trans \Rightarrow bool$ **where**
T $(Trans$ - - ou $s') =$
$(\exists$ $uid \in UIDs.$
$(\exists$ $cid.$ $PID \in\in paperIDs$ s' $cid \wedge isPC$ s' cid $uid \wedge$
$pref$ s' uid $PID \neq Conflict \wedge phase$ s' $cid \geq disPH)$
$\vee$
$(\exists$ $cid.$ $PID \in\in paperIDs$ s' $cid \wedge isPC$ s' cid $uid \wedge phase$ s' $cid \geq notifPH)$
$\vee$
$isAUT$ s' uid $PID \wedge (\exists$ $cid.$ $PID \in\in paperIDs$ s' $cid \wedge phase$ s' $cid \geq notifPH)$
)

declare $T.simps$ $[simp$ $del]$

definition B :: $value$ $list \Rightarrow value$ $list \Rightarrow bool$ **where**
B vl $vl1 \equiv vl \neq []$

interpretation *BD-Security-IO* **where**
$istate = istate$ **and** $step = step$ **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma *reachNT-non-isPC-isChair*:
assumes $reachNT$ s **and** $uid \in UIDs$
shows
$(PID \in\in paperIDs$ s $cid \wedge isPC$ s cid $uid \wedge phase$ s $cid \geq disPH$
$\longrightarrow pref$ s uid $PID = Conflict \wedge phase$ s $cid < notifPH) \wedge$
$(PID \in\in paperIDs$ s $cid \wedge isChair$ s cid $uid \wedge phase$ s $cid \geq disPH$
$\longrightarrow pref$ s uid $PID = Conflict \wedge phase$ s $cid < notifPH) \wedge$
$(isAut$ s cid uid $PID \longrightarrow phase$ s $cid < notifPH)$
$\langle proof \rangle$

lemma T-φ-γ:
assumes 1: $reachNT$ s **and** 2: $step$ s $a = (ou,s')$ φ $(Trans$ s a ou $s')$
shows $\neg$ γ $(Trans$ s a ou $s')$
$\langle proof \rangle$

lemma *eqExcPID-step-out*:
assumes $s's1'$: $eqExcPID$ s $s1$
and $step$: $step$ s $a = (ou,s')$ **and** $step1$: $step$ $s1$ $a = (ou1,s1')$
and sT: $reachNT$ s **and** $s1$: $reach$ $s1$
and PID: $PID \in\in paperIDs$ s cid

and *UIDs*: *userOfA a* $\in$ *UIDs*
shows *ou* = *ou1*
$\langle proof \rangle$

definition $\Delta 1$ :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$\Delta 1\ s\ vl\ s1\ vl1 \equiv$
$(\forall\ cid.\ PID \in\in paperIDs\ s\ cid \longrightarrow phase\ s\ cid < disPH) \wedge s = s1 \wedge B\ vl\ vl1$

definition $\Delta 2$ :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$\Delta 2\ s\ vl\ s1\ vl1 \equiv$
$\exists\ cid\ uid.$
$PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid = disPH \wedge$
$isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict \wedge$
$eqExcPID\ s\ s1$

definition $\Delta 3$ :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$\Delta 3\ s\ vl\ s1\ vl1 \equiv$
$\exists\ cid.\ PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH \wedge eqExcPID\ s\ s1 \wedge vl1 =$
$[]$

definition Δe :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$\Delta e\ s\ vl\ s1\ vl1 \equiv$
$vl \neq [] \wedge$
$((\exists\ cid.\ PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid > disPH)$
$\vee$
$(\exists\ cid.\ PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid \geq disPH \wedge$
$\neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict))$
$)$

lemma *istate*-$\Delta 1$:
assumes *B*: *B vl vl1*
shows $\Delta 1$ *istate vl istate vl1*
$\langle proof \rangle$

lemma *unwind-cont*-$\Delta 1$: *unwind-cont* $\Delta 1$ $\{\Delta 1, \Delta 2, \Delta e\}$
$\langle proof \rangle$

lemma *unwind-cont*-$\Delta 2$: *unwind-cont* $\Delta 2$ $\{\Delta 2, \Delta 3, \Delta e\}$
$\langle proof \rangle$

lemma *unwind-cont*-$\Delta 3$: *unwind-cont* $\Delta 3$ $\{\Delta 3, \Delta e\}$
$\langle proof \rangle$

definition *K1exit* **where**
$K1exit\ cid\ s \equiv$
$(PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid \geq disPH \wedge \neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge$
$pref\ s\ uid\ PID \neq Conflict))$

lemma *invarNT-K1exit*: *invarNT* (*K1exit cid*)
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
⟨*proof*⟩

definition *K2exit* **where**
K2exit cid s ≡ *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *disPH*

lemma *invarNT-K2exit*: *invarNT* (*K2exit cid*)
⟨*proof*⟩

lemma *noVal-K2exit*: *noVal* (*K2exit cid*) *v*
⟨*proof*⟩

lemma *unwind-exit-*Δ*e*: *unwind-exit* Δ*e*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Decision-All*
imports
Decision-NCPC
Decision-NCPC-Aut
begin

end
theory *Reviewer-Assignment-Intro*
imports ../*Safety-Properties*
begin

9 Reviewer Assignment Confidentiality

In this section, we prove confidentiality properties for the assignment of reviewers to a paper PID submitted to a conference.

The secrets (values) of interest are taken to be pairs (uid,Uids), where uid is a user and Uids is a set of users. The pairs arise from actions that appoint reviewers to the paper PID:

- uid is the appointed reviewer
- Uids is the set of PC members having no conflict with the paper

The use of the second component, which turns out to be the same for the entire sequence of values[3] is needed in order to express the piece of information (knowledge) that the appointed reviewers are among the non-conflicted PC members.[4]

Here, we have two points of compromise between the bound and the trigger (which yield two properties). Let

- T1 denote "PC membership having no conflict with that paper and the conference having moved to the reviewing phase"
- T2 denote "authorship of the paper and the conference having moved to the notification phase"

The two trigger-bound combinations are:

- weak trigger (T1 or T2) paired with strong bound (allowing to learn nothing beyond the public knowledge that the reviewers are among PC members having no conflict with that paper)
- strong trigger (T1) paired with weak bound (allowing to additionally learn the number of reviewers)

end

theory *Reviewer-Assignment-Value-Setup*
  imports *Reviewer-Assignment-Intro*
begin

9.1 Preliminaries

declare *updates-commute-paper*[*simp*]

consts *PID* :: *paperID*

definition *eqExcRLR* :: *role list* ⇒ *role list* ⇒ *bool* **where**
eqExcRLR rl rl1 ≡ [*r* ← *rl* . ¬ *isRevRoleFor PID r*] = [*r* ← *rl1* . ¬ *isRevRoleFor PID r*]

lemma *eqExcRLR-set*:
assumes *1*: *eqExcRLR rl rl1* **and** *2*: ¬ *isRevRoleFor PID r*
shows *r* ∈∈ *rl* ⟷ *r* ∈∈ *rl1*
⟨*proof*⟩

lemmas *eqExcRLR* = *eqExcRLR-def*

[3]This is because conflicts can no longer be changed at the time when reviewers can be appointed, i.e., in the reviewing phase.
[4]In CoCon, only PC members can be appointed as reviewers; there is no subreviewing facility.

lemma *eqExcRLR-eq*[*simp*,*intro*!]: *eqExcRLR rl rl*
⟨*proof*⟩

lemma *eqExcRLR-sym*:
assumes *eqExcRLR rl rl1*
shows *eqExcRLR rl1 rl*
⟨*proof*⟩

lemma *eqExcRLR-trans*:
assumes *eqExcRLR rl rl1* **and** *eqExcRLR rl1 rl2*
shows *eqExcRLR rl rl2*
⟨*proof*⟩

lemma *eqExcRLR-imp*:
assumes *s*: *reach s* **and** *pid*: $pid \neq PID$ **and**
1: *eqExcRLR* (*roles s cid uid*) (*roles s1 cid uid*)
shows
isRevNth s cid uid pid = *isRevNth s1 cid uid pid* ∧
isRev s cid uid pid = *isRev s1 cid uid pid* ∧
getRevRole s cid uid pid = *getRevRole s1 cid uid pid* ∧
getReviewIndex s cid uid pid = *getReviewIndex s1 cid uid pid* (**is** *?A* ∧ *?B* ∧ *?C* ∧ *?D*)
⟨*proof*⟩

lemma *eqExcRLR-imp2*:
assumes *eqExcRLR* (*roles s cid uid*) (*roles s1 cid uid*)
shows
isPC s cid uid = *isPC s1 cid uid* ∧
isChair s cid uid = *isChair s1 cid uid* ∧
isAut s cid uid = *isAut s1 cid uid*
⟨*proof*⟩

lemma *filter-eq-imp*:
assumes $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
and *filter Q xs* = *filter Q ys*
shows *filter P xs* = *filter P ys*
⟨*proof*⟩

lemma *arg-cong3*: $a = a1 \Longrightarrow b = b1 \Longrightarrow c = c1 \Longrightarrow h\ a\ b\ c = h\ a1\ b1\ c1$
⟨*proof*⟩

lemmas *map-concat-cong1* = *arg-cong*[**where** *f* = *concat*, *OF arg-cong2*[**where** *f* = *map*, *OF* - *refl*]]
lemmas *If-cong1* = *arg-cong3*[**where** *h* = *If*, *OF* - *refl refl*]

lemma *diff-cong1*: $a = a1 \Longrightarrow (a \neq b) = (a1 \neq b)$ ⟨*proof*⟩

lemma *isRev-pref-notConflict*:
assumes *reach s* **and** *isRev s cid uid pid*
shows *pref s uid pid* $\neq$ *Conflict*
$\langle proof \rangle$

lemma *isRev-pref-notConflict-isPC*:
assumes *reach s* **and** *isRev s cid uid pid*
shows *pref s uid pid* $\neq$ *Conflict* $\wedge$ *isPC s cid uid*
$\langle proof \rangle$

lemma *eqExcRLR-imp-isRevRole-imp*:
assumes *eqExcRLR rl rl1*
shows $[r \leftarrow rl.\ \neg\ isRevRole\ r] = [r \leftarrow rl1\ .\ \neg\ isRevRole\ r]$
$\langle proof \rangle$

lemma *notIsPC-eqExLRL-roles-eq*:
assumes *s*: *reach s* **and** *s1*: *reach s1* **and** *PID*: *PID* $\in\in$ *paperIDs s cid*
and *pc*: $\neg$ *isPC s cid uid*
and *eq*: *eqExcRLR* (*roles s cid uid*) (*roles* (*s1*::*state*) *cid uid*)
shows *roles s cid uid* = *roles s1 cid uid*
$\langle proof \rangle$

lemma *foo1*: $P\ a \Longrightarrow [r \leftarrow List.insert\ a\ l\ .\ P\ r] = (if\ a \in set\ l\ then\ filter\ P\ l\ else\ a\#filter\ P\ l)$
$\langle proof \rangle$

lemma *foo2*: $[\![eqExcRLR\ rl\ rl';\ \neg\ isRevRoleFor\ PID\ x]\!] \Longrightarrow eqExcRLR\ (List.insert\ x\ rl)\ (List.insert\ x\ rl')$
$\langle proof \rangle$

lemma *foo3*:
assumes *eqExcRLR rl rl′ isRevRoleFor PID x*
shows *eqExcRLR* (*List.insert x rl*) (*rl′*)
and *eqExcRLR* (*rl*) (*List.insert x rl′*)
$\langle proof \rangle$

The notion of two states being equal everywhere except on the reviewer roles for PID:

definition *eqExcPID* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *bool* **where**
eqExcPID s s1 $\equiv$
confIDs s = *confIDs s1* $\wedge$ *conf s* = *conf s1* $\wedge$
userIDs s = *userIDs s1* $\wedge$ *pass s* = *pass s1* $\wedge$ *user s* = *user s1*
$\wedge$
($\forall$ *cid uid*. *eqExcRLR* (*roles s cid uid*) (*roles s1 cid uid*))
$\wedge$
paperIDs s = *paperIDs s1*
$\wedge$
paper s = *paper s1*
$\wedge$

pref s = *pref s1* ∧
voronkov s = *voronkov s1* ∧
news s = *news s1* ∧ *phase s* = *phase s1*

lemma *eqExcPID-eq*[*simp*,*intro*!]: *eqExcPID s s*
⟨*proof*⟩

lemma *eqExcPID-sym*:
assumes *eqExcPID s s1* **shows** *eqExcPID s1 s*
⟨*proof*⟩

lemma *eqExcPID-trans*:
assumes *eqExcPID s s1* **and** *eqExcPID s1 s2* **shows** *eqExcPID s s2*
⟨*proof*⟩

lemma *eqExcPID-imp*:
eqExcPID s s1 ⟹
confIDs s = *confIDs s1* ∧ *conf s* = *conf s1* ∧
userIDs s = *userIDs s1* ∧ *pass s* = *pass s1* ∧ *user s* = *user s1*
∧
eqExcRLR (*roles s cid uid*) (*roles s1 cid uid*)
∧
paperIDs s = *paperIDs s1*
∧
paper s = *paper s1*
∧
pref s = *pref s1* ∧
voronkov s = *voronkov s1* ∧
news s = *news s1* ∧ *phase s* = *phase s1* ∧
getAllPaperIDs s = *getAllPaperIDs s1*
⟨*proof*⟩

lemma *eqExcPID-imp'*:
assumes *s*: *reach s* **and** *ss1*: *eqExcPID s s1* **and** *pid*: *pid* ≠ *PID* ∨ *PID* ≠ *pid*
shows
isRev s cid uid pid = *isRev s1 cid uid pid* ∧
getRevRole s cid uid pid = *getRevRole s1 cid uid pid* ∧
getReviewIndex s cid uid pid = *getReviewIndex s1 cid uid pid*
⟨*proof*⟩

lemma *eqExcPID-imp1*:
eqExcPID s s1 ⟹ *pid* ≠ *PID* ∨ *PID* ≠ *pid* ⟹
getNthReview s pid n = *getNthReview s1 pid n*
⟨*proof*⟩

lemma *eqExcPID-imp2*:
assumes *reach s* **and** *eqExcPID s s1* **and** *pid* ≠ *PID* ∨ *PID* ≠ *pid*

shows *getReviewersReviews s cid pid* = *getReviewersReviews s1 cid pid*
⟨*proof*⟩

lemma *eqExcPID-imp3*:
reach s $\Longrightarrow$ *eqExcPID s s1* $\Longrightarrow$ *pid* $\neq$ *PID* $\vee$ *PID* $\neq$ *pid*
$\Longrightarrow$
getNthReview s pid = *getNthReview s1 pid*
⟨*proof*⟩

lemma *eqExcPID-cong*[*simp*, *intro*]:
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*confIDs* := *uu1*|)) (*s1* (|*confIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*conf* := *uu1*|)) (*s1* (|*conf* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*userIDs* := *uu1*|)) (*s1* (|*userIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*pass* := *uu1*|)) (*s1* (|*pass* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*user* := *uu1*|)) (*s1* (|*user* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*roles* := *uu1*|)) (*s1* (|*roles* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*paperIDs* := *uu1*|)) (*s1* (|*paperIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*paper* := *uu1*|)) (*s1* (|*paper* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*pref* := *uu1*|)) (*s1* (|*pref* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*voronkov* := *uu1*|)) (*s1* (|*voronkov* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*news* := *uu1*|)) (*s1* (|*news* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID* (*s* (|*phase* := *uu1*|)) (*s1* (|*phase* := *uu2*|))

⟨*proof*⟩

A slightly weaker state equivalence that allows differences in the reviews of paper *PID*. It is used for the confidentiality property that doesn't cover the authors of that paper in the notification phase (when the authors will learn the contents of the reviews).

fun *eqExcR* :: *paper* ⇒ *paper* ⇒ *bool* **where**
eqExcR (*Paper name info ct reviews dis decs*)
  (*Paper name1 info1 ct1 reviews1 dis1 decs1*) =
(*name* = *name1* $\wedge$ *info* = *info1* $\wedge$ *ct* = *ct1* $\wedge$ *dis* = *dis1* $\wedge$ *decs* = *decs1*)

lemma *eqExcR*:
eqExcR pap pap1 =
(*titlePaper pap* = *titlePaper pap1* ∧ *abstractPaper pap* = *abstractPaper pap1* ∧
contentPaper pap = *contentPaper pap1* ∧
disPaper pap = *disPaper pap1* ∧ *decsPaper pap* = *decsPaper pap1*)
⟨*proof*⟩

lemma *eqExcR-eq*[*simp*,*intro*!]: *eqExcR pap pap*
⟨*proof*⟩

lemma *eqExcR-sym*:
assumes *eqExcR pap pap1*
shows *eqExcR pap1 pap*
⟨*proof*⟩

lemma *eqExcR-trans*:
assumes *eqExcR pap pap1* **and** *eqExcR pap1 pap2*
shows *eqExcR pap pap2*
⟨*proof*⟩

definition *eeqExcPID* **where**
eeqExcPID paps paps1 ≡
∀ *pid*. *if pid* = *PID then eqExcR* (*paps pid*) (*paps1 pid*) *else paps pid* = *paps1 pid*

lemma *eeqExcPID-eeq*[*simp*,*intro*!]: *eeqExcPID s s*
⟨*proof*⟩

lemma *eeqExcPID-sym*:
assumes *eeqExcPID s s1* **shows** *eeqExcPID s1 s*
⟨*proof*⟩

lemma *eeqExcPID-trans*:
assumes *eeqExcPID s s1* **and** *eeqExcPID s1 s2* **shows** *eeqExcPID s s2*
⟨*proof*⟩

lemma *eeqExcPID-imp*:
eeqExcPID paps paps1 ⟹ *eqExcR* (*paps PID*) (*paps1 PID*)
⟦*eeqExcPID paps paps1*; *pid* ≠ *PID*⟧ ⟹ *paps pid* = *paps1 pid*
⟨*proof*⟩

lemma *eeqExcPID-cong*:
assumes *eeqExcPID paps paps1*
and *pid* = *PID* ⟹ *eqExcR uu uu1*
and *pid* ≠ *PID* ⟹ *uu* = *uu1*
shows *eeqExcPID* (*paps* (*pid* := *uu*)) (*paps1*(*pid* := *uu1*))
⟨*proof*⟩

lemma *eeqExcPID-RDD*:

eeqExcPID paps paps1 $\Longrightarrow$
titlePaper (*paps PID*) = *titlePaper* (*paps1 PID*) ∧
abstractPaper (*paps PID*) = *abstractPaper* (*paps1 PID*) ∧
contentPaper (*paps PID*) = *contentPaper* (*paps1 PID*) ∧
disPaper (*paps PID*) = *disPaper* (*paps1 PID*) ∧
decsPaper (*paps PID*) = *decsPaper* (*paps1 PID*)
⟨*proof*⟩

definition *eqExcPID2* :: *state* ⇒ *state* ⇒ *bool* **where**
eqExcPID2 s s1 ≡
confIDs s = *confIDs s1* ∧ *conf s* = *conf s1* ∧
userIDs s = *userIDs s1* ∧ *pass s* = *pass s1* ∧ *user s* = *user s1*
∧
(∀ *cid uid*. *eqExcRLR* (*roles s cid uid*) (*roles s1 cid uid*))
∧
paperIDs s = *paperIDs s1*
∧
eeqExcPID (*paper s*) (*paper s1*)
∧
pref s = *pref s1* ∧
voronkov s = *voronkov s1* ∧
news s = *news s1* ∧ *phase s* = *phase s1*

lemma *eqExcPID2-eq*[*simp*,*intro*!]: *eqExcPID2 s s*
⟨*proof*⟩

lemma *eqExcPID2-sym*:
assumes *eqExcPID2 s s1* **shows** *eqExcPID2 s1 s*
⟨*proof*⟩

lemma *eqExcPID2-trans*:
assumes *eqExcPID2 s s1* **and** *eqExcPID2 s1 s2* **shows** *eqExcPID2 s s2*
⟨*proof*⟩

lemma *eqExcPID2-imp*:
eqExcPID2 s s1 $\Longrightarrow$
confIDs s = *confIDs s1* ∧ *conf s* = *conf s1* ∧
userIDs s = *userIDs s1* ∧ *pass s* = *pass s1* ∧ *user s* = *user s1*
∧
eqExcRLR (*roles s cid uid*) (*roles s1 cid uid*)
∧
paperIDs s = *paperIDs s1*
∧
eeqExcPID (*paper s*) (*paper s1*)
∧
pref s = *pref s1* ∧
voronkov s = *voronkov s1* ∧

$news\ s = news\ s1 \wedge phase\ s = phase\ s1 \wedge$

$getAllPaperIDs\ s = getAllPaperIDs\ s1$
$\langle proof \rangle$

lemma *eeqExcPID-imp2*:
assumes *pid*: $pid \neq PID$ **and**
1: *eeqExcPID* (*paper s*) (*paper s1*)
shows
$reviewsPaper\ (paper\ s\ pid) = reviewsPaper\ (paper\ s1\ pid)$
$\langle proof \rangle$

lemma *eqExcPID2-imp′*:
assumes *s*: *reach s* **and** *ss1*: *eqExcPID2 s s1* **and** *pid*: $pid \neq PID \vee PID \neq pid$
shows
$isRev\ s\ cid\ uid\ pid = isRev\ s1\ cid\ uid\ pid \wedge$
$getRevRole\ s\ cid\ uid\ pid = getRevRole\ s1\ cid\ uid\ pid \wedge$
$getReviewIndex\ s\ cid\ uid\ pid = getReviewIndex\ s1\ cid\ uid\ pid \wedge$
$reviewsPaper\ (paper\ s\ pid) = reviewsPaper\ (paper\ s1\ pid)$
$\langle proof \rangle$

lemma *eqExcPID2-imp1*:
$eqExcPID2\ s\ s1 \Longrightarrow eqExcR\ (paper\ s\ pid)\ (paper\ s1\ pid)$
$eqExcPID2\ s\ s1 \Longrightarrow pid \neq PID \vee PID \neq pid \Longrightarrow$
$paper\ s\ pid = paper\ s1\ pid \wedge$
$getNthReview\ s\ pid\ n = getNthReview\ s1\ pid\ n$
$\langle proof \rangle$

lemma *eqExcPID2-imp2*:
assumes *reach s* **and** *eqExcPID2 s s1* **and** $pid \neq PID \vee PID \neq pid$
shows $getReviewersReviews\ s\ cid\ pid = getReviewersReviews\ s1\ cid\ pid$
$\langle proof \rangle$

lemma *eqExcPID2-imp3*:
$reach\ s \Longrightarrow eqExcPID2\ s\ s1 \Longrightarrow pid \neq PID \vee PID \neq pid$
$\Longrightarrow$
$getNthReview\ s\ pid = getNthReview\ s1\ pid$
$\langle proof \rangle$

lemma *eqExcPID2-RDD*:
$eqExcPID2\ s\ s1 \Longrightarrow$
$titlePaper\ (paper\ s\ PID) = titlePaper\ (paper\ s1\ PID) \wedge$
$abstractPaper\ (paper\ s\ PID) = abstractPaper\ (paper\ s1\ PID) \wedge$
$contentPaper\ (paper\ s\ PID) = contentPaper\ (paper\ s1\ PID) \wedge$
$disPaper\ (paper\ s\ PID) = disPaper\ (paper\ s1\ PID) \wedge$
$decsPaper\ (paper\ s\ PID) = decsPaper\ (paper\ s1\ PID)$
$\langle proof \rangle$

lemma *eqExcPID2-cong*[*simp*, *intro*]:
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*confIDs* := *uu1*|)) (*s1* (|*confIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*conf* := *uu1*|)) (*s1* (|*conf* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*userIDs* := *uu1*|)) (*s1* (|*userIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*pass* := *uu1*|)) (*s1* (|*pass* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*user* := *uu1*|)) (*s1* (|*user* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*roles* := *uu1*|)) (*s1* (|*roles* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*paperIDs* := *uu1*|)) (*s1* (|*paperIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *eeqExcPID uu1 uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*paper* := *uu1*|)) (*s1* (|*paper* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*pref* := *uu1*|)) (*s1* (|*pref* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*voronkov* := *uu1*|)) (*s1* (|*voronkov* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*news* := *uu1*|)) (*s1* (|*news* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqExcPID2 s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqExcPID2* (*s* (|*phase* := *uu1*|)) (*s1* (|*phase* := *uu2*|))

⟨*proof*⟩

lemma *eqExcPID2-Paper*:
assumes *s′s1′*: *eqExcPID2 s s1*
and *paper s pid* = *Paper title abstract content reviews dis decs*
and *paper s1 pid* = *Paper title1 abstract1 content1 reviews1 dis1 decs1*
shows *title* = *title1* ∧ *abstract* = *abstract1* ∧ *content* = *content1* ∧ *dis* = *dis1* ∧ *decs* = *decs1*
⟨*proof*⟩

lemma *cReview-step-eqExcPID2*:
assumes *a*:
a = *Cact* (*cReview cid uid p PID uid′*)
and *step s a* = (*ou*,*s′*)
shows *eqExcPID2 s s′*
⟨*proof*⟩

9.2 Value Setup

type-synonym $value = userID \times userID\ set$

fun φ :: $(state, act, out)\ trans \Rightarrow bool$ **where**
$\varphi\ (Trans\ \text{-}\ (Cact\ (cReview\ cid\ uid\ p\ pid\ uid'))\ ou\ \text{-}) =$
$(pid = PID \wedge ou = outOK)$
|
$\varphi\ \text{-} = False$

fun f :: $(state, act, out)\ trans \Rightarrow value$ **where**
$f\ (Trans\ s\ (Cact\ (cReview\ cid\ uid\ p\ pid\ uid'))\ \text{-}\ s') =$
$(uid', \{uid'.\ isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\})$

lemma φ-*def2*:
$\varphi\ (Trans\ s\ a\ ou\ s') =$
$(ou = outOK \wedge$
$(\exists\ cid\ uid\ p\ uid'.\ a = Cact\ (cReview\ cid\ uid\ p\ PID\ uid')))$
$\langle proof \rangle$

fun χ :: $act \Rightarrow bool$ **where**
$\chi\ (Uact\ (uReview\ cid\ uid\ p\ pid\ n\ rc)) = (pid = PID)$
|
$\chi\ (UUact\ (uuReview\ cid\ uid\ p\ pid\ n\ rc)) = (pid = PID)$
|
$\chi\ \text{-} = False$

lemma χ-*def2*:
$\chi\ a =$
$(\exists\ cid\ uid\ p\ n\ rc.\ a = Uact\ (uReview\ cid\ uid\ p\ PID\ n\ rc) \vee$
$a = UUact\ (uuReview\ cid\ uid\ p\ PID\ n\ rc))$
$\langle proof \rangle$

lemma *eqExcPID-step-*φ*-imp*:
assumes *s*: *reach s* **and** *ss1*: *eqExcPID s s1*

and *PID*: $PID \in\in paperIDs\ s\ cid$ **and** *ph*: $phase\ s\ cid > revPH$

and *step*: $step\ s\ a = (ou, s')$ **and** *step1*: $step\ s1\ a = (ou1, s1')$
and φ: $\neg\ \varphi\ (Trans\ s\ a\ ou\ s')$
shows $\neg\ \varphi\ (Trans\ s1\ a\ ou1\ s1')$
$\langle proof \rangle$

lemma *eqExcPID-step-*φ:
assumes *reach s* **and** *reach s1* **and** *ss1*: *eqExcPID s s1*

and *PID*: $PID \in\in paperIDs\ s\ cid$ **and** *ph*: $phase\ s\ cid > revPH$

and *step*: $step\ s\ a = (ou, s')$ **and** *step1*: $step\ s1\ a = (ou1, s1')$

shows φ (*Trans s a ou s'*) = φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *non-eqExcPID-step-φ-imp*:
assumes *s*: *reach s* **and** *ss1*: *eqExcPID s s1*
and *PID*: *PID* $\in\in$ *paperIDs s cid* **and** *ou*: *ou* $\neq$ *outErr*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and φ: $\neg$ φ (*Trans s a ou s'*)
shows $\neg$ φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqExcPID-step*:
assumes *s*: *reach s* **and** *s1*: *reach s1*
and *ss1*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*)
and *step1*: *step s1 a* = (*ou1*,*s1'*)
and *PID*: *PID* $\in\in$ *paperIDs s cid*
and *ou-ph*: *ou* $\neq$ *outErr* $\vee$ *phase s cid* > *revPH*
and φ: $\neg$ φ (*Trans s a ou s'*) **and** χ: $\neg$ χ *a*
shows *eqExcPID s' s1'*
⟨*proof*⟩

lemma *φ-step-eqExcPID2*:
assumes φ: φ (*Trans s a ou s'*)
and *s*: *step s a* = (*ou*,*s'*)
shows *eqExcPID2 s s'*
⟨*proof*⟩

lemma *eqExcPID2-step*:
assumes *s*: *reach s*
and *ss1*: *eqExcPID2 s s1*
and *step*: *step s a* = (*ou*,*s'*)
and *step1*: *step s1 a* = (*ou1*,*s1'*)
and *PID*: *PID* $\in\in$ *paperIDs s cid* **and** *ph*: *phase s cid* $\geq$ *revPH*
and φ: $\neg$ φ (*Trans s a ou s'*)
shows *eqExcPID2 s' s1'*
⟨*proof*⟩

lemma *eqExcPID2-step-φ-imp*:
assumes *s*: *reach s* **and** *ss1*: *eqExcPID2 s s1*

and *PID*: *PID* $\in\in$ *paperIDs s cid* **and** *ph*: *phase s cid* > *revPH*

and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and φ: $\neg$ φ (*Trans s a ou s'*)
shows $\neg$ φ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *eqExcPID2-step-φ*:
assumes *reach s* **and** *reach s1* **and** *ss1*: *eqExcPID2 s s1*

and *PID*: $PID \in\in$ *paperIDs s cid* **and** *ph*: *phase s cid* $>$ *revPH*

and *step*: *step s a* $= (ou,s')$ **and** *step1*: *step s1 a* $= (ou1,s1')$
shows φ (*Trans s a ou s'*) $= \varphi$ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

lemma *non-eqExcPID2-step-φ-imp*:
assumes *s*: *reach s* **and** *ss1*: *eqExcPID2 s s1*
and *PID*: $PID \in\in$ *paperIDs s cid* **and** *ou*: $ou \neq outErr$
and *step*: *step s a* $= (ou,s')$ **and** *step1*: *step s1 a* $= (ou1,s1')$
and φ: $\neg\ \varphi$ (*Trans s a ou s'*)
shows $\neg\ \varphi$ (*Trans s1 a ou1 s1'*)
⟨*proof*⟩

end
theory *Reviewer-Assignment-NCPC*
imports *../Observation-Setup Reviewer-Assignment-Value-Setup Bounded-Deducibility-Security.Compositiona*
begin

9.3 Confidentiality protection from non-PC-members

We verify the following property:

A group of users UIDs learn nothing about the reviewers assigned to a paper PID except for their number and the fact that they are PC members having no conflict with that paper unless/until the user becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the reviewing phase.

fun *T* :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
T (*Trans - - ou s'*) =
(∃ *uid* ∈ *UIDs*. ∃ *cid*.
PID ∈∈ *paperIDs s' cid* ∧ *isPC s' cid uid* ∧ *pref s' uid PID* ≠ *Conflict* ∧ *phase s' cid* ≥ *revPH*)

term *isAUT*

declare *T.simps* [*simp del*]

definition B :: *value list* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
$B\ vl\ vl1 \equiv$
$vl \neq [] \wedge$
$length\ vl = length\ vl1 \wedge$
$distinct\ (map\ fst\ vl1) \wedge fst\ `\ (set\ vl1) \subseteq snd\ (hd\ vl) \wedge snd\ `\ (set\ vl1) = \{snd\ (hd\ vl)\}$

interpretation *BD-Security-IO* **where**
$istate = istate$ **and** $step = step$ **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma *reachNT-non-isPC-isChair*:
assumes $reachNT\ s$ **and** $uid \in UIDs$
shows
$(PID \in\in paperIDs\ s\ cid \wedge isPC\ s\ cid\ uid \longrightarrow$
$pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH) \wedge$
$(PID \in\in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \longrightarrow$
$pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH)$
$\langle proof \rangle$

lemma *T-φ-γ*:
assumes *1*: $reachNT\ s$ **and** *2*: $step\ s\ a = (ou,s')\ \varphi\ (Trans\ s\ a\ ou\ s')$
shows $\neg\ \gamma\ (Trans\ s\ a\ ou\ s')$
$\langle proof \rangle$

lemma *T-φ-γ-stronger*:
assumes *s*: $reach\ s$ **and** *0*: $PID \in\in paperIDs\ s\ cid$
and *2*: $step\ s\ a = (ou,s')\ \varphi\ (Trans\ s\ a\ ou\ s')$
and *1*: $\forall\ uid \in UIDs.\ isChair\ s\ cid\ uid \longrightarrow pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH$
shows $\neg\ \gamma\ (Trans\ s\ a\ ou\ s')$
$\langle proof \rangle$

lemma *T-φ-γ-1*:
assumes *s*: $reachNT\ s$ **and** *s1*: $reach\ s1$ **and** *PID*: $PID \in\in paperIDs\ s\ cid$
and *ss1*: $eqExcPID\ s\ s1$
and *step1*: $step\ s1\ a = (ou1,s1')$ **and** *φ1*: $\varphi\ (Trans\ s1\ a\ ou1\ s1')$
and *φ*: $\neg\ \varphi\ (Trans\ s\ a\ ou\ s')$
shows $\neg\ \gamma\ (Trans\ s1\ a\ ou1\ s1')$
$\langle proof \rangle$

lemma *notIsPCorConflict-eqExcPID-roles-eq*:
assumes *s*: $reach\ s$ **and** *s1*: $reach\ s1$ **and** *PID*: $PID \in\in paperIDs\ s\ cid$
and *pc*: $\neg\ isPC\ s\ cid\ uid \vee pref\ s\ uid\ PID = Conflict$
and *eeq*: $eqExcPID\ s\ s1$

shows *roles s cid uid = roles s1 cid uid*
⟨*proof*⟩

lemma *notInPaperIDs-eqExcPID-roles-eq*:
assumes *s*: *reach s* **and** *s1*: *reach s1* **and** *PID*: ¬ *PID* ∈∈ *paperIDs s cid*
and *eq*: *eqExcPID s s1*
shows *roles s cid uid = roles s1 cid uid*
⟨*proof*⟩

lemma *eqExcPID-step-out*:
assumes *ss1*: *eqExcPID s s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *sT*: *reachNT s* **and** *s1*: *reach s1*
and *PID*: *PID* ∈∈ *paperIDs s cid* **and** *ph*: *phase s cid* ≥ *revPH*
and φ: ¬ φ (*Trans s a ou s'*) **and** φ*1*: ¬ φ (*Trans s1 a ou1 s1'*) **and** χ: ¬ χ *a*
and *UIDs*: *userOfA a* ∈ *UIDs*
shows *ou = ou1*
⟨*proof*⟩

lemma *eqExcPID-step-φ-eqExcPID-out*:
assumes *s*: *reach s* **and** *s1*: *reach s1*
and *a*: *a = Cact* (*cReview cid uid p PID uid'*)
and *a1*: *a1 = Cact* (*cReview cid uid p PID uid1'*)
and *ss1*: *eqExcPID s s1* **and** *step*: *step s a* = (*outOK*,*s'*)
and *pc*: *isPC s cid uid1'* ∧ *pref s uid1' PID* ≠ *Conflict*
and *rv1*: ¬ *isRev s1 cid uid1' PID* **and** *step1*: *step s1 a1* = (*ou1*,*s1'*)
shows *eqExcPID s' s1'* ∧ *ou1 = outOK*
⟨*proof*⟩

lemma *eqExcPID-ex-isNthReview*:
assumes *s*: *reach s* **and** *s1*: *reach s1* **and** *e*: *eqExcPID s s1*
and *i*: *isRevNth s cid uid PID n*
shows ∃ *uid1*. *isRevNth s1 cid uid1 PID n*
⟨*proof*⟩

lemma *eqExcPID-step-χ1*:
assumes *s*: *reach s* **and** *s1*: *reach s1*
and *a*: *a = Uact* (*uReview cid uid p PID n rc*)
and *ss1*: *eqExcPID s s1* **and** *step*: *step s a* = (*outOK*,*s'*)
shows
∃ *s1' uid1 p*.
  isRevNth s1 cid uid1 PID n ∧
  step s1 (*Uact* (*uReview cid uid1 p PID n rc*)) = (*outOK*,*s1'*) ∧
  eqExcPID s' s1'
⟨*proof*⟩

lemma *eqExcPID-step-χ2*:
assumes *s*: *reach s* **and** *s1*: *reach s1*
and *a*: *a = UUact* (*uuReview cid uid p PID n rc*)

and *ss1*: *eqExcPID s s1* **and** *step*: *step s a* = (*outOK*,*s'*)
shows
$\exists$ *s1' uid1 p*.
  isRevNth s1 cid uid1 PID n $\wedge$
  step s1 (*UUact* (*uuReview cid uid1 p PID n rc*)) = (*outOK*,*s1'*) $\wedge$
  eqExcPID s' s1'
$\langle$*proof*$\rangle$

definition Δ*1* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*1 s vl s1 vl1* $\equiv$
($\forall$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\longrightarrow$ *phase s cid* < *revPH*) $\wedge$ *s* = *s1*
$\wedge$ *B vl vl1*

definition Δ*2* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*2 s vl s1 vl1* $\equiv$
$\exists$ *cid uid*.
  PID $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* = *revPH* $\wedge$
  isChair s cid uid $\wedge$ *pref s uid PID* $\neq$ *Conflict* $\wedge$
  eqExcPID s s1 $\wedge$
  length vl = *length vl1* $\wedge$
  distinct (*map fst vl1*) $\wedge$
  fst ‘ (*set vl1*) $\subseteq$ {*uid'*. *isPC s cid uid'* $\wedge$ *pref s uid' PID* $\neq$ *Conflict*} $\wedge$
  fst ‘ (*set vl1*) $\cap$ {*uid'*. *isRev s1 cid uid' PID*} = {} $\wedge$
  snd ‘ (*set vl1*) $\subseteq$ {{*uid'*. *isPC s cid uid'* $\wedge$ *pref s uid' PID* $\neq$ *Conflict*}}

definition Δ*3* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*3 s vl s1 vl1* $\equiv$
$\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* > *revPH* $\wedge$ *eqExcPID s s1* $\wedge$ *vl1* =
[]

definition Δ*e* :: *state* $\Rightarrow$ *value list* $\Rightarrow$ *state* $\Rightarrow$ *value list* $\Rightarrow$ *bool* **where**
Δ*e s vl s1 vl1* $\equiv$
vl $\neq$ [] $\wedge$
(
 ($\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *revPH* $\wedge$
    $\neg$ ($\exists$ *uid*. *isChair s cid uid* $\wedge$ *pref s uid PID* $\neq$ *Conflict*))
 $\vee$
 ($\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *revPH* $\wedge$
    snd (*hd vl*) $\neq$ {*uid'*. *isPC s cid uid'* $\wedge$ *pref s uid' PID* $\neq$ *Conflict*})
 $\vee$
 ($\exists$ *cid*. *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* > *revPH*)
)

lemma *istate*-Δ*1*:
assumes *B*: *B vl vl1*
shows Δ*1 istate vl istate vl1*
$\langle$*proof*$\rangle$

lemma *unwind-cont-*Δ*1*: *unwind-cont* $\Delta 1$ $\{\Delta 1, \Delta 2, \Delta e\}$
⟨*proof*⟩

lemma *not-*φ*-isRev-isPC-persists*:
assumes *reach s*
PID $\in\in$ *paperIDs s cid* **and** $\neg$ φ (*Trans s a ou s'*)
and *step s a* = (*ou*,*s'*)
shows *isRev s' cid uid PID* = *isRev s cid uid PID* $\wedge$ *isPC s' cid uid* = *isPC s cid uid*
⟨*proof*⟩

lemma γ*-not*χ*-eqButPID-outErr*:
assumes *sT*: *reachNT s* **and** *s1*: *reach s1*
and *UIDs*: *userOfA a* $\in$ *UIDs* **and** *step*: *step s a* = (*outErr*,*s'*)
and *ss1*: *eqExcPID s s1* **and** *PID*: *PID* $\in\in$ *paperIDs s CID*
shows *step s1 a* = (*outErr*,*s1*)
⟨*proof*⟩

lemma *exists-failedAct*:
$\exists$ *a*. *step s a* = (*outErr*,*s*) $\wedge$ *userOfA a* = *uid*
⟨*proof*⟩

lemma *unwind-cont-*Δ*2*: *unwind-cont* $\Delta 2$ $\{\Delta 2, \Delta 3, \Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*Δ*3*: *unwind-cont* $\Delta 3$ $\{\Delta 3, \Delta e\}$
⟨*proof*⟩

definition *K1exit* **where**
K1exit cid s $\equiv$ *PID* $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *revPH* $\wedge$
$\neg$ ($\exists$ *uid*. *isChair s cid uid* $\wedge$ *pref s uid PID* $\neq$ *Conflict*)

lemma *invarNT-K1exit*: *invarNT* (*K1exit cid*)
⟨*proof*⟩

lemma *noVal-K1exit*: *noVal* (*K1exit cid*) *v*
⟨*proof*⟩

definition *K2exit* **where**
K2exit cid s v $\equiv$
PID $\in\in$ *paperIDs s cid* $\wedge$ *phase s cid* $\geq$ *revPH* $\wedge$
snd v $\neq$ $\{$*uid'*. *isPC s cid uid'* $\wedge$ *pref s uid' PID* $\neq$ *Conflict*$\}$

lemma *revPH-isPC-constant*:
assumes *s*: *reach s*
and *step s a* = (*ou*,*s'*)
and *pid* $\in\in$ *paperIDs s cid* **and** *phase s cid* $\geq$ *revPH*

shows *isPC* s' *cid* uid' = *isPC s cid* uid'
⟨*proof*⟩

lemma *revPH-pref-constant*:
assumes *s*: *reach s*
and *step s a* = (*ou*,s')
and *pid* ∈∈ *paperIDs s cid* **and** *phase s cid* ≥ *revPH*
shows *pref* s' *uid pid* = *pref s uid pid*
⟨*proof*⟩

lemma *invarNT-K2exit*: *invarNT* (λ *s*. *K2exit cid s v*)
⟨*proof*⟩

lemma *noVal-K2exit*: *noVal2* (*K2exit cid*) *v*
⟨*proof*⟩

definition *K3exit* **where**
K3exit cid s ≡ *PID* ∈∈ *paperIDs s cid* ∧ *phase s cid* > *revPH*

lemma *invarNT-K3exit*: *invarNT* (*K3exit cid*)
⟨*proof*⟩

lemma *noVal-K3exit*: *noVal* (*K3exit cid*) *v*
⟨*proof*⟩

lemma *unwind-exit-*Δ*e*: *unwind-exit* Δ*e*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Reviewer-Assignment-NCPC-Aut*
imports ../*Observation-Setup Reviewer-Assignment-Value-Setup Bounded-Deducibility-Security.Compositiona*
begin

9.4 Confidentiality protection from users who are not PC members or authors of the paper

We verify the following property:

A group of users UIDs learn nothing about the reviewers assigned to a paper PID except for the fact that they are PC members having no conflict with that paper unless/until one of the following occurs:

- the user becomes a PC member in the paper's conference having no conflict with that paper and the conference moves to the reviewing phase, or

- the user becomes an author of the paper and the conference moves to the notification phase.

fun T :: $(state,act,out)$ $trans \Rightarrow bool$ **where**
T $(Trans$ - - ou $s') =$
$(\exists\ uid \in UIDs.$
$(\exists\ cid.\ PID \in\in paperIDs\ s'\ cid \wedge isPC\ s'\ cid\ uid \wedge pref\ s'\ uid\ PID \neq Conflict \wedge phase\ s'\ cid \geq revPH)$
$\vee$
$(\exists\ cid.\ PID \in\in paperIDs\ s'\ cid \wedge isAut\ s'\ cid\ uid\ PID \wedge phase\ s'\ cid \geq notifPH)$
$)$

declare $T.simps$ $[simp\ del]$

definition B :: $value\ list \Rightarrow value\ list \Rightarrow bool$ **where**
$B\ vl\ vl1 \equiv$
$vl \neq [] \wedge$
$distinct\ (map\ fst\ vl1) \wedge fst\ `\ (set\ vl1) \subseteq snd\ (hd\ vl) \wedge snd\ `\ (set\ vl1) = \{snd\ (hd\ vl)\}$

interpretation $BD\text{-}Security\text{-}IO$ **where**
$istate = istate$ **and** $step = step$ **and**
$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
$\langle proof \rangle$

lemma $reachNT\text{-}non\text{-}isPC\text{-}isChair$:
assumes $reachNT\ s$ **and** $uid \in UIDs$
shows
$(PID \in\in paperIDs\ s\ cid \wedge isPC\ s\ cid\ uid \longrightarrow pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH)$
$\wedge$
$(PID \in\in paperIDs\ s\ cid \wedge isChair\ s\ cid\ uid \longrightarrow pref\ s\ uid\ PID = Conflict \vee phase\ s\ cid < revPH)$
$\wedge$
$(PID \in\in paperIDs\ s\ cid \wedge isAut\ s\ cid\ uid\ PID \longrightarrow$
$phase\ s\ cid < notifPH)$
$\langle proof \rangle$

lemma $T\text{-}\varphi\text{-}\gamma$:
assumes 1: $reachNT\ s$ **and** 2: $step\ s\ a = (ou,s')\ \varphi\ (Trans\ s\ a\ ou\ s')$
shows $\neg\ \gamma\ (Trans\ s\ a\ ou\ s')$
$\langle proof \rangle$

lemma *T-φ-γ-stronger*:
assumes *s*: *reach s* **and** *0*: *PID ∈∈ paperIDs s cid*
and *2*: *step s a = (ou,s′) φ (Trans s a ou s′)*
and *1*: *∀ uid ∈ UIDs. isChair s cid uid ⟶ pref s uid PID = Conflict ∨ phase s cid < revPH*
shows *¬ γ (Trans s a ou s′)*
⟨*proof*⟩

lemma *T-φ-γ-1*:
assumes *s*: *reachNT s* **and** *s1*: *reach s1* **and** *PID*: *PID ∈∈ paperIDs s cid*
and *ss1*: *eqExcPID2 s s1*
and *step1*: *step s1 a = (ou1,s1′)* **and** *φ1*: *φ (Trans s1 a ou1 s1′)*
and *φ*: *¬ φ (Trans s a ou s′)*
shows *¬ γ (Trans s1 a ou1 s1′)*
⟨*proof*⟩

lemma *notInPaperIDs-eqExLRL-roles-eq*:
assumes *s*: *reach s* **and** *s1*: *reach s1* **and** *PID*: *¬ PID ∈∈ paperIDs s cid*
and *eq*: *eqExcPID2 s s1*
shows *roles s cid uid = roles s1 cid uid*
⟨*proof*⟩

lemma *eqExcPID2-step-out*:
assumes *ss1*: *eqExcPID2 s s1*
and *step*: *step s a = (ou,s′)* **and** *step1*: *step s1 a = (ou1,s1′)*
and *sT*: *reachNT s* **and** *s1*: *reach s1*
and *PID*: *PID ∈∈ paperIDs s cid* **and** *ph*: *phase s cid ≥ revPH*
and *φ*: *¬ φ (Trans s a ou s′)* **and** *φ1*: *¬ φ (Trans s1 a ou1 s1′)*
and *UIDs*: *userOfA a ∈ UIDs*
shows *ou = ou1*
⟨*proof*⟩

definition *Δ1* :: *state ⇒ value list ⇒ state ⇒ value list ⇒ bool* **where**
Δ1 s vl s1 vl1 ≡
(∀ cid. PID ∈∈ paperIDs s cid ⟶ phase s cid < revPH) ∧ s = s1
∧ B vl vl1

definition *Δ2* :: *state ⇒ value list ⇒ state ⇒ value list ⇒ bool* **where**
Δ2 s vl s1 vl1 ≡
∃ cid uid.
PID ∈∈ paperIDs s cid ∧ phase s cid = revPH ∧
isChair s cid uid ∧ pref s uid PID ≠ Conflict ∧
eqExcPID2 s s1 ∧
distinct (map fst vl1) ∧
fst ‘ (set vl1) ⊆ {uid′. isPC s cid uid′ ∧ pref s uid′ PID ≠ Conflict} ∧
fst ‘ (set vl1) ∩ {uid′. isRev s1 cid uid′ PID} = {} ∧
snd ‘ (set vl1) ⊆ {{uid′. isPC s cid uid′ ∧ pref s uid′ PID ≠ Conflict}}

definition $\Delta 3$ :: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 3\ s\ vl\ s1\ vl1 \equiv$
$\exists\ cid.\ PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH \wedge eqExcPID2\ s\ s1 \wedge vl1 =$
$[]$

definition Δe :: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta e\ s\ vl\ s1\ vl1 \equiv$
$vl \neq [] \wedge$
$($
$(\exists\ cid.\ PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$
$\neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict))$
$\vee$
$(\exists\ cid.\ PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$
$snd\ (hd\ vl) \neq \{uid'.\ isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\})$
$\vee$
$(\exists\ cid.\ PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH)$
$)$

lemma *istate-*$\Delta 1$:
assumes B: $B\ vl\ vl1$
shows $\Delta 1\ istate\ vl\ istate\ vl1$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: $unwind\text{-}cont\ \Delta 1\ \{\Delta 1, \Delta 2, \Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 2$: $unwind\text{-}cont\ \Delta 2\ \{\Delta 2, \Delta 3, \Delta e\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 3$: $unwind\text{-}cont\ \Delta 3\ \{\Delta 3, \Delta e\}$
⟨*proof*⟩

definition *K1exit* **where**
$K1exit\ cid\ s \equiv PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$
$\neg (\exists\ uid.\ isChair\ s\ cid\ uid \wedge pref\ s\ uid\ PID \neq Conflict)$

lemma *invarNT-K1exit*: $invarNT\ (K1exit\ cid)$
⟨*proof*⟩

lemma *noVal-K1exit*: $noVal\ (K1exit\ cid)\ v$
⟨*proof*⟩

definition *K2exit* **where**
$K2exit\ cid\ s\ v \equiv$
$PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid \geq revPH \wedge$

$snd\ v \neq \{uid'.\ isPC\ s\ cid\ uid' \wedge pref\ s\ uid'\ PID \neq Conflict\}$

lemma *revPH-isPC-constant*:
assumes *s*: *reach s*
and $step\ s\ a = (ou,s')$
and $pid \in\in paperIDs\ s\ cid$ **and** $phase\ s\ cid \geq revPH$
shows $isPC\ s'\ cid\ uid' = isPC\ s\ cid\ uid'$
⟨*proof*⟩

lemma *revPH-pref-constant*:
assumes *s*: *reach s*
and $step\ s\ a = (ou,s')$
and $pid \in\in paperIDs\ s\ cid$ **and** $phase\ s\ cid \geq revPH$
shows $pref\ s'\ uid\ pid = pref\ s\ uid\ pid$
⟨*proof*⟩

lemma *invarNT-K2exit*: $invarNT\ (\lambda\ s.\ K2exit\ cid\ s\ v)$
⟨*proof*⟩

lemma *noVal-K2exit*: $noVal2\ (K2exit\ cid)\ v$
⟨*proof*⟩

definition *K3exit* **where**
$K3exit\ cid\ s \equiv PID \in\in paperIDs\ s\ cid \wedge phase\ s\ cid > revPH$

lemma *invarNT-K3exit*: $invarNT\ (K3exit\ cid)$
⟨*proof*⟩

lemma *noVal-K3exit*: $noVal\ (K3exit\ cid)\ v$
⟨*proof*⟩

lemma *unwind-exit-*Δe: $unwind\text{-}exit\ \Delta e$
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end
theory *Reviewer-Assignment-All*
imports
Reviewer-Assignment-NCPC
Reviewer-Assignment-NCPC-Aut
begin

end
theory *Traceback-Properties*
imports *Safety-Properties*

begin

10 Traceback properties

In this section, we prove various traceback properties, by essentially giving trace-based justifications of certain occurring situations that are relevant for access to information:

Being an author. If a user is an author of a paper, then either the user has registered the paper in the first place or, inductively, has been appointed as coauthor by another author.

Being a chair. If a user is a chair of a conference, then either that user has registered the conference which has been approved by the superuser or, inductively, that user has been appointed by an existing chair of that conference.

Being a PC member. If a user is a PC member in a conference, then the user either must have been the original chair or must have been appointed by a chair.

Being a reviewer. If a user is a paper's reviewer, then the user must have been appointed by a chair (from among the PC members who have not declared a conflict with the paper).

Having conflict. If a user has conflict with a paper, then the user is either an author of the paper or the conflict has been declared by that user or by a paper's author, in such a way that between the moment when the conflict has been last declared and the current moment there is no transition that successfully removes the conflict.

Conference reaching a phase. If a conference is in a given phase different from "no phase", then this has happened as a consequence of either a conference approval action by the superuser (if the phase is Setup) or a phase change action by a chair (otherwise).

More details and explanations can be found in [6, Section 3.6].

10.1 Preliminaries

inductive *trace-between* :: *state* ⇒ (*state*,*act*,*out*) *trans trace* ⇒ *state* ⇒ *bool*
where
empty[*simp*]: *trace-between s* [] *s*
| *step*: ⟦*trace-between s tr sh*; *step sh a* = (*ou*,*s'*)⟧ ⟹ *trace-between s* (*tr*@[*Trans sh a ou s'*]) *s'*

inductive-simps
trace-ft-empty[*simp*]: *trace-between s* [] s' **and**
trace-ft-snoc: *trace-between s* (*tr*@[*trn*]) s'
thm *trace-ft-empty trace-ft-snoc*

lemma *trace-ft-append*: *trace-between s* (*tr1*@*tr2*) s'
$\longleftrightarrow$ ($\exists$ *sh*. *trace-between s tr1 sh* $\wedge$ *trace-between sh tr2* s')
⟨*proof*⟩

lemma *trace-ft-Cons*: *trace-between s* (*trn*#*tr*) s'
$\longleftrightarrow$ ($\exists$ *sh ou a*. *trn* = *Trans s a ou sh* $\wedge$ *step s a* = (*ou*,*sh*) $\wedge$ *trace-between sh tr* s')
⟨*proof*⟩

lemmas *trace-ft-simps* = *trace-ft-empty trace-ft-snoc trace-ft-Cons trace-ft-append*

inductive *trace-to* :: (*state*,*act*,*out*) *trans trace* $\Rightarrow$ *state* $\Rightarrow$ *bool* **where**
empty: *trace-to* [] *istate*
| *step*: ⟦*trace-to tr s*; *step s a* = (*ou*,s')⟧ $\Longrightarrow$ *trace-to* (*tr*@[*Trans s a ou* s']) s'

lemma *trace-to-ft*: *trace-to tr s* $\longleftrightarrow$ *trace-between istate tr s*
⟨*proof*⟩

inductive-simps *trace-to-empty*[*simp*]: *trace-to* [] *s*

lemma *trace-to-reach*: **assumes** *trace-to tr s* **shows** *reach s*
⟨*proof*⟩

lemma *reach-to-trace*: **assumes** *reach s* **obtains** *tr* **where** *trace-to tr s*
⟨*proof*⟩

lemma *reach-trace-to-conv*: *reach s* $\longleftrightarrow$ ($\exists$ *tr*. *trace-to tr s*)
⟨*proof*⟩

thm *trace-to.induct*[*no-vars*]

lemma *trace-to-induct*[*case-names empty step*, *induct set*]:
⟦*trace-to x1 x2*; *P* [] *istate*;
$\bigwedge$*tr s a ou* s'.
⟦*trace-to tr s*; *P tr s*; *reach s*; *reach* s'; *step s a* = (*ou*, s')⟧
$\Longrightarrow$ *P* (*tr* ## *Trans s a ou* s') s'⟧
$\Longrightarrow$ *P x1 x2*
⟨*proof*⟩

10.2 Authorship

Only the creator of a paper, and users explicitly added by other authors, are authors of a paper.

inductive *isAut'* :: (*state*,*act*,*out*) *trans trace* $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *paperID* $\Rightarrow$

bool **where**
creator: $\llbracket$ *trn* = *Trans* - (*Cact* (*cPaper cid uid* - *pid* - -)) *outOK* - $\rrbracket$
$\Longrightarrow$ *isAut′* (*tr*@[*trn*]) *cid uid pid*

| *co-author*: $\llbracket$
isAut′ tr cid uid′ pid;
trn = *Trans* - (*Cact* (*cAuthor cid uid′* - *pid uid*)) *outOK* - $\rrbracket$
$\Longrightarrow$ *isAut′* (*tr*@[*trn*]) *cid uid pid*

| *irrelevant*: *isAut′ tr cid uid′ pid* $\Longrightarrow$ *isAut′* (*tr*@[-]) *cid uid′ pid*

lemma *justify-author*:
assumes *trace-to tr s*
assumes *isAut s cid uid pid*
shows *isAut′ tr cid uid pid*
⟨*proof*⟩

lemma *author-justify*:
assumes *trace-to tr s*
assumes *isAut′ tr cid uid pid*
shows *isAut s cid uid pid*
⟨*proof*⟩

theorem *isAut-eq*: *trace-to tr s* $\Longrightarrow$ *isAut s cid uid pid* $\longleftrightarrow$ *isAut′ tr cid uid pid*

⟨*proof*⟩

10.3 Becoming a Conference Chair

inductive *isChair′* :: (*state*,*act*,*out*) *trans trace* $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *bool* **where**
creator: $\llbracket$ *trn*=*Trans* - (*Cact* (*cConf cid uid* - - -)) *outOK* - $\rrbracket$
$\Longrightarrow$ *isChair′* (*tr*@[*trn*]) *cid uid*
| *add-chair*: $\llbracket$ *isChair′ tr cid uid′*; *trn* = *Trans* - (*Cact* (*cChair cid uid′* - *uid*)) *outOK* - $\rrbracket$
$\Longrightarrow$ *isChair′* (*tr*@[*trn*]) *cid uid*
| *irrelevant*: $\llbracket$*isChair′ tr cid uid*$\rrbracket$ $\Longrightarrow$ *isChair′* (*tr*@[-]) *cid uid*

lemma *justify-chair*:
assumes *trace-to tr s*
assumes *isChair s cid uid*
shows *isChair′ tr cid uid*
⟨*proof*⟩

lemma *chair-justify*:
assumes *trace-to tr s*
assumes *isChair′ tr cid uid*
shows *isChair s cid uid*
⟨*proof*⟩

theorem *isChair-eq*: *trace-to tr s* $\Longrightarrow$ *isChair s cid uid* = *isChair' tr cid uid*

⟨*proof*⟩

10.4 Committee Membership

inductive *isPC'* :: (*state*,*act*,*out*) *trans trace* $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *bool* **where**
chair: *isChair' tr cid uid* $\Longrightarrow$ *isPC' tr cid uid*
| *add-com*: ⟦ *isChair' tr cid uid'*; *trn* = *Trans* - (*Cact* (*cPC cid uid'* - *uid*)) *outOK* - ⟧
$\Longrightarrow$ *isPC'* (*tr*@[*trn*]) *cid uid*
| *irrelevant*: ⟦*isPC' tr cid uid*⟧ $\Longrightarrow$ *isPC'* (*tr*@[-]) *cid uid*

lemma *justify-com*:
assumes *trace-to tr s*
assumes *isPC s cid uid*
shows *isPC' tr cid uid*
⟨*proof*⟩

lemma *com-justify*:
assumes *trace-to tr s*
assumes *isPC' tr cid uid*
shows *isPC s cid uid*
⟨*proof*⟩

theorem *isPC-eq*: *trace-to tr s* $\Longrightarrow$ *isPC s cid uid* = *isPC' tr cid uid*

⟨*proof*⟩

10.5 Being a Reviewer

inductive *isRev'* :: (*state*,*act*,*out*) *trans trace* $\Rightarrow$ *confID* $\Rightarrow$ *userID* $\Rightarrow$ *paperID* $\Rightarrow$ *bool* **where**
add-rev: ⟦ *isChair' tr cid uid'*; *trn* = *Trans* - (*Cact* (*cReview cid uid'* - *pid uid*)) *outOK* - ⟧
$\Longrightarrow$ *isRev'* (*tr*@[*trn*]) *cid uid pid*
| *irrelevant*: ⟦*isRev' tr cid uid pid*⟧ $\Longrightarrow$ *isRev'* (*tr*@[-]) *cid uid pid*

lemma *justify-rev*:
assumes *trace-to tr s*
assumes *isRev s cid uid pid*
shows *isRev' tr cid uid pid*
⟨*proof*⟩

lemma *rev-justify*:
assumes *trace-to tr s*
assumes *isRev' tr cid uid pid*
shows *isRev s cid uid pid*
⟨*proof*⟩

theorem *isRev-eq*: $trace\text{-}to\ tr\ s \Longrightarrow isRev\ s\ cid\ uid\ pid = isRev'\ tr\ cid\ uid\ pid$

$\langle proof \rangle$

10.6 Conflicts

fun *irrev-conflict* :: $userID \Rightarrow paperID \Rightarrow (state,act,out)\ trans \Rightarrow bool$

where
$irrev\text{-}conflict\ uid\ pid\ (Trans\ \text{-}\ (Cact\ (cPaper\ \text{-}\ uid'\ \text{-}\ pid'\ \text{-}\ \text{-}))\ outOK\ \text{-})$
$\longleftrightarrow uid'=uid \wedge pid'=pid$
$|\ irrev\text{-}conflict\ uid\ pid\ (Trans\ \text{-}\ (Cact\ (cAuthor\ \text{-}\ \text{-}\ \text{-}\ pid'\ uid'))\ outOK\ \text{-})$
$\longleftrightarrow uid'=uid \wedge pid'=pid$
$|\ irrev\text{-}conflict\ uid\ pid\ \text{-} \longleftrightarrow False$

fun *set-conflict* :: $userID \Rightarrow paperID \Rightarrow (state,act,out)\ trans \Rightarrow bool$

where
$set\text{-}conflict\ uid\ pid\ (Trans\ \text{-}\ (Cact\ (cConflict\ \text{-}\ \text{-}\ \text{-}\ pid'\ uid'))\ outOK\ \text{-})$
$\longleftrightarrow uid'=uid \wedge pid'=pid$
$|\ set\text{-}conflict\ uid\ pid\ (Trans\ \text{-}\ (Uact\ (uPref\ \text{-}\ uid'\ \text{-}\ pid'\ Conflict))\ outOK\ \text{-})$
$\longleftrightarrow uid'=uid \wedge pid'=pid$
$|\ set\text{-}conflict\ \text{-}\ \text{-}\ \text{-} \longleftrightarrow False$

fun *reset-conflict* :: $userID \Rightarrow paperID \Rightarrow (state,act,out)trans \Rightarrow bool$

where
$reset\text{-}conflict\ uid\ pid\ (Trans\ \text{-}\ (Uact\ (uPref\ \text{-}\ uid'\ \text{-}\ pid'\ pr))\ outOK\ \text{-})$
$\longleftrightarrow uid'=uid \wedge pid'=pid \wedge pr \neq Conflict$
$|\ reset\text{-}conflict\ \text{-}\ \text{-}\ \text{-} \longleftrightarrow False$

definition *conflict-trace* :: $userID \Rightarrow paperID \Rightarrow (state,act,out)\ trans\ trace \Rightarrow bool$

where
$conflict\text{-}trace\ uid\ pid\ tr \equiv$
$(\exists\, trn \in set\ tr.\ irrev\text{-}conflict\ uid\ pid\ trn)$
$\vee\ (\exists\, tr1\ trn\ tr2.\ tr=tr1 @ trn \# tr2\ \wedge$
$set\text{-}conflict\ uid\ pid\ trn \wedge (\forall\, trn \in set\ tr2.\ \neg reset\text{-}conflict\ uid\ pid\ trn))$

lemma *irrev-conflict-impl-author*:
assumes $trace\text{-}to\ tr\ s$
assumes $\exists\, trn \in set\ tr.\ irrev\text{-}conflict\ uid\ pid\ trn$
shows $\exists\, cid.\ isAut\ s\ cid\ uid\ pid$
$\langle proof \rangle$

lemma *irrev-conflict-impl-conflict*:
assumes $trace\text{-}to\ tr\ s$
assumes $\exists\, trn \in set\ tr.\ irrev\text{-}conflict\ uid\ pid\ trn$

shows *pref s uid pid* = *Conflict*
⟨*proof*⟩

lemma *conflict-justify*:
assumes *TR*: *trace-to tr s*
assumes *conflict-trace uid pid tr*
shows *pref s uid pid* = *Conflict*
⟨*proof*⟩

lemma *justify-conflict*:
assumes *TR*: *trace-to tr s*
assumes *pref s uid pid* = *Conflict*
shows *conflict-trace uid pid tr*
⟨*proof*⟩

theorem *conflict-eq*:
assumes *trace-to tr s*
shows *pref s uid pid* = *Conflict* ⟷ *conflict-trace uid pid tr*
⟨*proof*⟩

10.7 Conference Phases

fun *is-uPhase* **where**
is-uPhase cid (*Trans* - (*Uact* (*uConfA cid′* - -)) *outOK* -) ⟷ *cid′*=*cid*
| *is-uPhase cid* (*Trans* - (*Uact* (*uPhase cid′* - - -)) *outOK* -) ⟷ *cid′*=*cid*
| *is-uPhase* - - ⟷ *False*

inductive *phase′* :: (*state*,*act*,*out*) *trans trace* ⇒ *confID* ⇒ *nat* ⇒ *bool* **where**
initial: *phase′* [] *cid noPH*
| *approve*: ⟦*phase′ tr cid noPH*; *trn*=*Trans s* (*Uact* (*uConfA cid* (*voronkov s*) -)) *outOK* - ⟧
⟹ *phase′* (*tr*@[*trn*]) *cid setPH*
| *advance*: ⟦*trn* = (*Trans* - (*Uact* (*uPhase cid uid* - *ph*)) *outOK* -); *isChair′ tr cid uid*⟧
⟹ *phase′* (*tr*@[*trn*]) *cid ph*
| *irrelevant*: ⟦*phase′ tr cid ph*; ¬*is-uPhase cid trn* ⟧ ⟹ *phase′* (*tr*@[*trn*]) *cid ph*

lemma *justify-phase*:
assumes *trace-to tr s*
assumes *phase s cid* = *ph*
shows *phase′ tr cid ph*
⟨*proof*⟩

lemma *phase-justify*:
assumes *trace-to tr s*
assumes *phase′ tr cid ph*
shows *phase s cid* = *ph*
⟨*proof*⟩

theorem *phase-eq*:
assumes *trace-to tr s*
shows *phase s cid* = *ph* $\longleftrightarrow$ *phase′ tr cid ph*
⟨*proof*⟩

end
theory *All-BD-Security-Instances-for-CoCon*
imports

Paper-Confidentiality/Paper-All

Review-Confidentiality/Review-All

Discussion-Confidentiality/Discussion-All

Decision-Confidentiality/Decision-All

Reviewer-Assignment-Confidentiality/Reviewer-Assignment-All

Traceback-Properties
begin

end

References

[1] T. Bauereiß, A. Pesenti Gritti, A. Popescu, and F. Raimondi. Cosmed: A confidentiality-verified social media platform. In J. C. Blanchette and S. Merz, editors, *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, volume 9807 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 2016.

[2] T. Bauereiß, A. Pesenti Gritti, A. Popescu, and F. Raimondi. Cosmed: A confidentiality-verified social media platform. *J. Autom. Reason.*, 61(1-4):113–139, 2018.

[3] S. Kanav, P. Lammich, and A. Popescu. A conference management system with verified document confidentiality. In A. Biere and R. Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014,*

Vienna, Austria, July 18-22, 2014. Proceedings, volume 8559 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2014.

[4] A. Popescu, T. Bauereiss, and P. Lammich. Bounded-Deducibility security (invited paper). In L. Cohen and C. Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPIcs*, pages 3:1–3:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[5] A. Popescu, P. Lammich, and T. Bauereiss. Bounded-deducibility security. In G. Klein, T. Nipkow, and L. Paulson, editors, *Archive of Formal Proofs*, 2014.

[6] A. Popescu, P. Lammich, and P. Hou. Cocon: A conference management system with formally verified document confidentiality. *J. Autom. Reason.*, 65(2):321–356, 2021.