Inference-proof View Update Transactions with Minimal Refusals

Joachim Biskup and Cornelia Tadros

Faculty of Computer Science
Information Systems and Security – ISSI

16. September 2011

technische universität dortmund

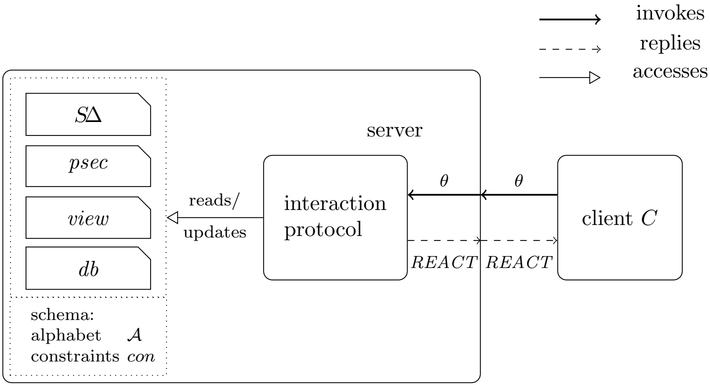
Objectives:

- **Confidentiality**: Protect confidential information in database instance db according to personalized confidentiality policy $psec$.
- **Information Sharing**: Provide database client with the following services:
  - access database view $view$,
  - query about (propositional) database instance,
  - update view with translation to update of database instance.
- **Inference Control**: Prevent Client to infer confidential information from query answers and update notifications.

Achievements: Inference-proof interaction protocols which automatically refuse some requests

- ensuring *confidentiality* and *database integrity*,
- providing the client with *no misinformation*,
- optimizing availability of view-updates under a policy of *last-minute intervention*.

Client-Server Interactions



θ query/view update request

Database component

- *propositional complete database instance*:
  - a set db of propositional variables.
  - defines truth-value assignment under closed world assumption (CWA).
- *database schema*:
  - alphabet $\mathcal{A}$ of propositional variables and
    integrity constraints $con \subseteq \mathcal{L}_{pl}^{\mathcal{A}}$ (set of propositional formulas).
  - defines reasonable instances db:
    $db \subseteq \mathcal{A}$
    $db \models con$ (with propositional model-of operator $\models$)
- *database view*:
  - information of the instance db that is visible to the client.
  - a set $view \subseteq \mathcal{L}_{pl}^{\mathcal{A}}$ such that $db \models view$ (No Misinformation).

Database Operations

Query:

- Syntax: $que(\phi)$ where $\phi \in \mathcal{L}_{pl}^{\mathcal{A}}$
- Semantics: evaluated by a database instance db
  eval$(\phi)(db) := (db \models \phi)$

View-update transactions:

- Syntax: $vtr(L)$ where L is a list of literals $\langle \mathcal{X}_1, \ldots, \mathcal{X}_l \rangle$ over pairwise distinct variables.
- Semantics: translated to the instance, complying with the *ACID* principles

$\neg a \in L$ delete variable a from db,

$a \in L$ insert variable a into db.

Ordinary View Update Processing

Client requests: $vtr(\langle \mathcal{X}_1, \ldots, \mathcal{X}_l \rangle)$
Server processes:

- Compute outstanding updates
  $Inc\Delta := \{\mathcal{X} \in L \mid \mathsf{eval}(\mathcal{X})(db) = false\}$.
- Compute modified instance
  $db^{Inc\Delta} := \underbrace{\{x \in db \mid \neg x \notin Inc\Delta\}}_{\text{deletions}} \cup \underbrace{(Inc\Delta \cap \mathcal{A})}_{\text{insertions}}$.
- Enforce integrity constraints
  If $db^{Inc\Delta} \not\models con$, undo modifications and notify client about integrity violation; else
- Update view
  $view := \underbrace{\mathsf{neg}(view, Inc\Delta)}_{\text{refreshed}} \cup \underbrace{\{\mathcal{X}_1, \ldots, \mathcal{X}_l\}}_{\text{requested}} \cup \underbrace{con}_{\text{integrity preservation}}$

View Refreshment

Refreshment in order to

- adjust outdated information (No Misinformation)
- preserve information content of the view (No Loss of Information)

Refreshing $view$ by neg$(view, Inc\Delta)$ achieves these properties:

- Each formula $\phi \in view$ is refreshed by neg$(\phi, Inc\Delta)$:
  Replace each occurence of a modified variable x in ϕ by $\neg x$.
- The refreshed formula is valid in the modified instance:

$$\text{eval}(\text{neg}(\phi, Inc\Delta))(db^{Inc\Delta}) \overset{(\star)}{=} \text{eval}(\phi)(db) = true$$

(($\star$) Lemma Negation Equivalence)

Example (Ordinary View-Update Processing)

Schema: $\mathcal{A} := \{a, b, c\}$ $\quad con := \{\neg c \Rightarrow a\}$

Instance: $db_1 := \{a, c\}$ $\quad$ with CWA, i.e., $db_1 = \{a, \neg b, c\}$

View: $view_1 := con \cup \{a \vee b, c\}$

Client requests: $vtr(\langle \neg c \rangle)$

Server processes:

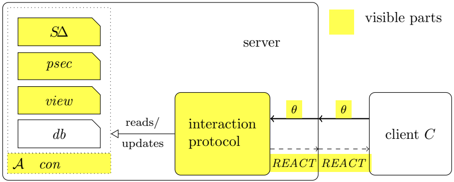
- Outstanding updates: $Inc\Delta = \{\neg c\}$
- Modify instance: $db_1^{Inc\Delta} = \{a, \neg b, \neg c\}$
- Enforce integrity : $db_1^{Inc\Delta} \models con$
- Update view: $view_2 = \mathsf{neg}(view_1, \{\neg c\}) \cup \{\neg c\} \cup con$

$$= \{\neg\neg c \Rightarrow a, a \vee b, \neg c\} \cup con$$

$$\equiv \{\neg c, a\} \quad (\equiv \text{logical equivalence})$$

Assumptions About the Client

- Unlimited computing power.
- Visible parts:
  - procedure of each interaction protocol P,
  - all server components except for db,
  - the outputs $REACT_i$, $view_i$ and $S\Delta_i$ of protocol P.



Policy Declaration

- Personalized confidentiality policy $psec$:
  - two disjoint sets $psec(TCP)$ and $psec(CCP)$ of propositional formulas,
  - declared by security administrator when creating the client's account.
- $\psi \in psec(TCP)$:
  - *potential secret* with **temporary confidentiality** requirement.
  - prohibits client to know that ψ is valid in the *current instance*.
  - may stand for, e.g., "Smith's phone number is 1234", "Smith's bank account number is xyz".
- $\psi \in psec(CCP)$:
  - potential secret with **continuous confidentiality** requirement.
  - prohibits client to know that ψ is valid in *some preceding or the current instance*.
  - may stand for, e.g.: "Smith has cancer".

Definition (Confidentiality Preservation by Protocol P)

Given a potential secret $\psi \in psec = psec(TCP) \uplus psec(CCP)$ and *admissible*, initial components con, db_0 and $view_0$, after a finite sequence Q of query and view update requests,

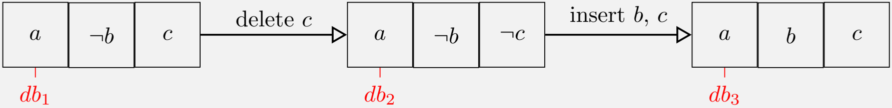
1. the client cannot distinguish the actual current instance db_Q from an alternative current instance db_Q^S, i.e, $\nu^C(P(con, db_0, psec, view_0, Q)) = \nu^C(P(con, db_0^S, psec, view_0, Q))$,
2. such that, if ψ requires temporary preservation, then ψ is not valid in db_Q^S,
3. if ψ requires continuous preservation, then ψ is not valid in db_Q^S and all preceding instances.

Inferences about Preceding Instances (1)

To reason about preceding instances the client must keep track of *effective updates*.

Example (Effective Updates)

The client successfully requested $\langle vtr(\neg c), vtr(b, c)\rangle$:

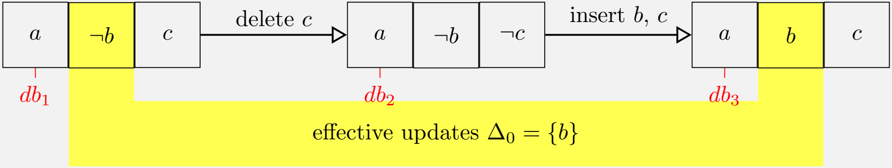


Inferences about Preceding Instances (1)

To reason about preceding instances the client must keep track of *effective updates*.

Example (Effective Updates)

The client successfully requested $\langle vtr(\neg c), vtr(b, c)\rangle$:

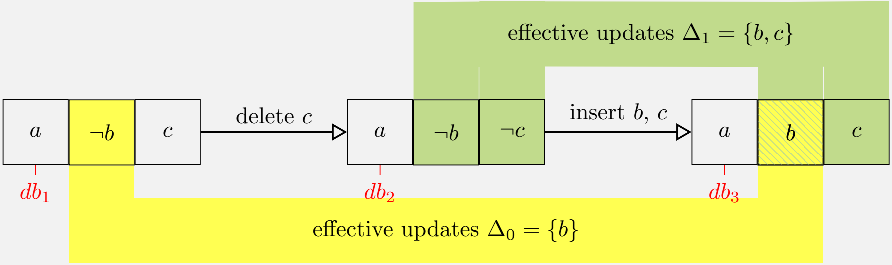


Inferences about Preceding Instances (1)

To reason about preceding instances the client must keep track of *effective updates*.

Example (Effective Updates)

The client successfully requested $\langle vtr(\neg c), vtr(b, c)\rangle$:

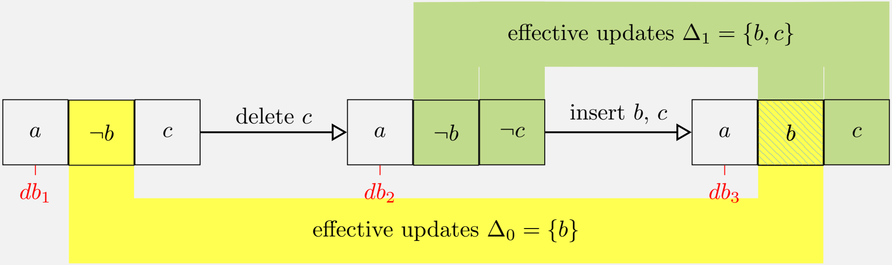


Inferences about Preceding Instances (1)

To reason about preceding instances the client must keep track of *effective updates*.

Example (Effective Updates)

The client successfully requested $\langle vtr(\neg c), vtr(b, c)\rangle$:



Effective Updates: $\langle\{b\}, \{b, c\}\rangle$.

Inferences about Preceding Instances (2)

After the successful view update $vtr(\langle \neg c \rangle)$, the client reasons:

- Is the potential secret $a \wedge c$ (with CCP) valid in the preceding instance db_1?
  - $a \wedge c$ is valid in db_1 iff $a \wedge \neg c$ is valid in db_2:

$$\begin{aligned} \text{eval}(a \wedge c)(db_1) &= \text{eval}(\text{neg}(a \wedge c, \{\neg c\}))(db_1^{\{\neg c\}}) \\ &= \text{eval}(a \wedge \neg c)(db_2) \text{ (Effective update } \neg c \text{ \& Lemma)} \end{aligned}$$

  - $a \wedge \neg c$ is valid in db_2, because c has been deleted and constraint $\neg c \Rightarrow a$ is preserved:

$$view_2 \supset \{\neg c, \neg c \Rightarrow a\} \vdash a \wedge \neg c \text{ (View \& propositional entailment } \vdash \text{)}$$

  - Consequently, $a \wedge c$ is valid in db_1.
- $psec(CCP) = \{a \wedge c\}$ is violated

Enforcing Continuous Confidentiality

After an interaction sequence
with j effective updates $\langle \Delta_0, \ldots, \Delta_{j-1} \rangle = S\Delta$
for each potential secret $\psi \in psec(CCP)$

- ψ is valid in the preceding instance after the $i+1$-th modification iff $\mathsf{neg}(\psi, \Delta_i)$ is valid in the current instance db.
- ψ is valid in some preceding instance iff $\mathsf{neg}(\psi, \Delta_0) \vee \ldots \vee \mathsf{neg}(\psi, \Delta_{j-1})$ is valid in the current instance db.
- Consequently, an interaction protocol must enforce the invariant:

$$\begin{aligned} view \not\models\ & \mathsf{neg}(\psi, \Delta_0) \vee \ldots \vee \mathsf{neg}(\psi, \Delta_{j-1}) \vee \psi \\ =\ & \mathsf{ccp}(\psi, S\Delta). \end{aligned}$$

Given the view and the effective updates, the client cannot reason that ψ previously has held or currently holds.

Inference-proof View-Update Transaction Protocol

Request: $vtr(\langle \mathcal{X}_1, \ldots, \mathcal{X}_l \rangle)$

1. **Outstanding Updates**: $Inc\Delta = \{\mathcal{X}_j \mid \text{eval}(\mathcal{X}_j)(db) = false\}$
   - Submit the query requests $que(\mathcal{X}_1), \ldots, que(\mathcal{X}_l)$ to the protocol for inference-proof query processing.
   - If one query request is refused, abort the transaction.
2. **Truthful View - Confidentiality Conflict**:
   - Check if updated view breaches confidentiality.
   - In case of a breach, abort the transaction.
3. **Integrity - Confidentiality Conflict**:
   - Check if notification of integrity violation conflicts confidentiality.
   - In case of a conflict, abort the transaction
   - else perform the integrity check.
4. **Ordinary Processing**:
   In case of integrity preservation, modify the instance and update the view.

We abbreviate $con\_conj := \bigwedge_{\phi \in con} \phi$

Case 3 Integrity - Confidentiality Conflict

1: **if** $view_{i-1}$ and $Inc\Delta$ disclose the result of integrity check **then**
2: continue ordinary processing accordingly,
3: **else if** $view_{i-1} \cup \text{neg}(\neg con\_conj, Inc\Delta) \vdash \text{ccp}(\psi, S\Delta_{i-1})$
for a $\psi \in psec(CCP)$
or
$view_{i-1} \cup \text{neg}(\neg con\_conj, Inc\Delta) \vdash \psi$
for a $\psi \in psec(TCP)$ **then**
4: **return** $REACT_i$:=*integrity check conflicts confidentiality* (exit)
5: **else**
6: Check integrity

$\text{neg}(\neg con\_conj, Inc\Delta)$	integrity violated after update $Inc\Delta$
$\text{ccp}(\psi, S\Delta_{i-1})$	given effective updates $S\Delta_{i-1}$, ψ has held or holds

Theorem (Confidentiality Preservation)

The query evaluation protocol and the view-update transaction protocol together preserve continuous and temporary confidentiality preservation.

Example (Inference-proof View-Update Processing)

We review our running example with request $vtr(\langle\neg c\rangle)$ but policy $psec(CCP) := \{\neg a \wedge b\}$ and $psec(TCP) := \emptyset$.

case	inference checks	computations
2	Truthful View - Confidentiality Conflict	
	neg$(view_1, Inc\Delta) \cup con = view_2$ $\equiv \{\neg c, a\} \not\vdash \neg a \wedge b \equiv$ ccp$(\neg a \wedge b, \overline{S\Delta})$	$\overline{S\Delta} = \langle\{\neg c\}\rangle$
3	Integrity - Confidentiality Conflict	
	passes until line 2 (result of integrity check not known to client); $view_1 \cup$ neg$(\neg con_conj, \{\neg c\})$ $= \{a \vee b, c, \neg c \Rightarrow a\} \cup$ neg$(\neg c \wedge \neg a, \{\neg c\})$ $\equiv \{c, \neg a, b\} \vdash \neg a \wedge b \equiv$ ccp$(\neg a \wedge b, \overline{S\Delta})$	

Above inference checks independent from database instance, so that request refused on any instance.

Availability Analysis

Availability policy: *last-minute intervention*

- intervene (i.e., refuse request) only if necessary for confidentiality,
- depending on $view$,
- respecting the client's immediate information needs.

Local optimality result:
there is no view-update protocol with certain properties, e.g.,

- regarding the view maintenance (no misinformation, no loss of information etc.),
- database integrity and confidentiality preservation, etc.

that in a one-step view update transaction

- exits with a successful update refused by our protocol, or
- provides the client with more (required) information.

Achievements:

- protocols for processing query and view update request by a single client to a complete propositional database instance,
- inference-proofness of these protocols with temporary or continuous confidentiality requirements,
- availability analysis with a local optimal result under an availability policy of last-minute intervention.

Related Work:

- Inference-proof view updates admitting misinformation in the view:
  - [Biskup et al] dynamic view update and refreshment protocols with lying
  - cover stories in MLS databases, cf. [Gabillon]
- Optimizing availability by preprocessing:
  - [Biskup, Wiese] inference-proof instance with minimal lies & personalized availability policy
  - [Dawson et al] lowest classification of data in MLS databases
  - [Ciriani et al] minimal vertical fragmentation at schema level with visibility constraints

Future Work

- Inference-proof refreshment protocol for multiple clients
- Implementation in existing prototype for controlled interaction execution
- Other database models, e.g., relational or incomplete
- Other temporal confidentiality requirements
- Comparing availability between refusal and “lying” approaches
- Non deterministic protocols

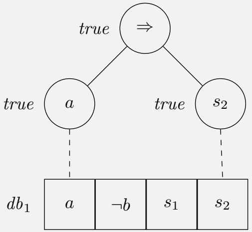
Thank you for your attention!

Appendix

Negation Equivalence

Example (Refreshment)

View update transaction $vtr(\langle \neg a, b \rangle)$ with $view = \{a \Rightarrow s_2\}$ on db_1

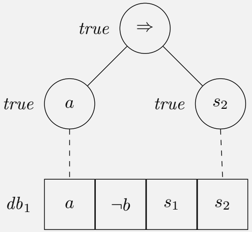


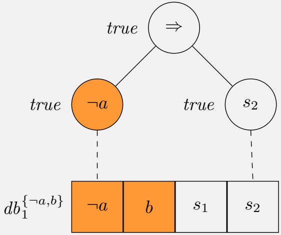
No misinformation in $view$

Negation Equivalence

Example (Refreshment)

View update transaction $vtr(\langle \neg a, b \rangle)$ with $view = \{a \Rightarrow s_2\}$ on db_1





No misinformation in $view$

No misinformation, no loss of information in refreshed view $\{\neg a \Rightarrow s_2\}$

Confidentiality Preservation

Sketch of Proof:

- Both protocols ensure the invariants:

$$view \not\models \mathsf{ccp}(\psi, S\Delta) \qquad \text{for each } \psi \in psec(CCP)$$
$$view \not\models \psi \qquad \text{for each } \psi \in psec(TCP)$$

- Based on these invariants, alternative sequences of instances are constructed that are indistinguishable and safe under the respective requirement. $\square$

- Availability policy: *last-minute intervention*
  - intervene (i.e., refuse request) only if necessary for confidentiality,
  - depending on $view$,
  - respecting the client's immediate information needs.
- In the following: Availability analysis of an one-step view-update transaction (Local Optimality).
- Assumption: client queries $que(\mathcal{X}_1), \ldots, que(\mathcal{X}_l)$ before request $vtr(\langle \mathcal{X}_1, \ldots, \mathcal{X}_l \rangle)$ without a refusal so that

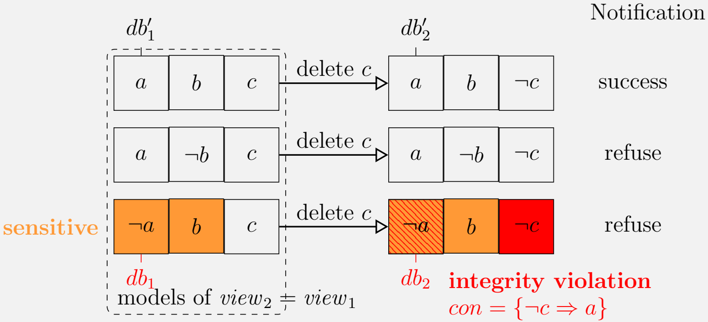
$$\underbrace{\{\neg\mathcal{X} \mid \mathcal{X} \in Inc\Delta\}}_{\text{outstanding}} \cup \underbrace{(\{\mathcal{X}_1, \ldots, \mathcal{X}_l\} \setminus Inc\Delta)}_{\text{void updates}} \subseteq view_{i-1}.$$

Definition (Proper Truthful Deterministic Protocol P)

- **Deterministic**, **Atomicity and Integrity** (ACID)
- **No Misinformation**: $db_i \models view_i$.
- **No Loss of Information**:
  Failed update: $view_i \vdash view_{i-1}$
  Successful update: $view_i \vdash \mathsf{neg}(view_{i-1}, Inc\Delta)$.
- **Cooperativeness**:
  Failed update: $view_{i-1} \cup \mathsf{neg}(\neg con\_conj, Inc\Delta) \vdash view_i$
  Successful update: $\mathsf{neg}(view_{i-1}, Inc\Delta) \cup con \vdash view_i$.
  $REACT$: truthful report about success/failure.
- **Soundness of Client View**: If for db'_{i-1} admissible the client observes different output given db_{i-1} and db'_{i-1}, then
  Failed update: $db'_{i-1} \not\models view_i$
  Successful update: $db'^{Inc\Delta}_{i-1} \not\models view_i$.
- **Confidentiality**

Example (Soundness of Client View)

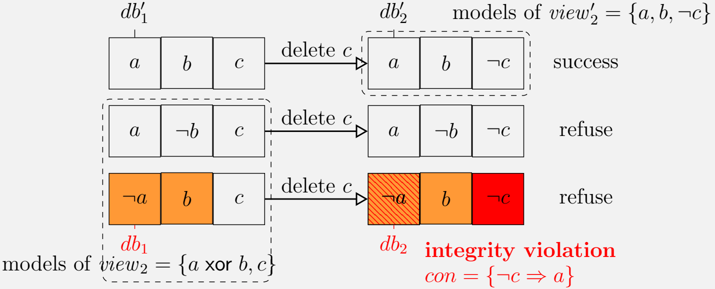
Reconsider the running example with **another protocol** P'



- Integrity conflicts policy $psec(CCP) = \{\neg a \wedge b\}$: refuse for db_1.
- Additional refusal due client's reasoning about processing of P' (*meta-inference*).
- Client distinguishes db'_1 and db_1 by the observed "*refuse*".
- Soundness of client view not ensured for db_1: $db'_1 \models view_2 = view_1$.

Example (Cooperativeness)

Another protocol P'' processes $vtr(\langle \neg c \rangle)$



- Sound of client view is ensured by $view_2$ and $view'_2$
- Cooperativeness is ensured for db_1:
  $view_1 \cup \text{neg}(\neg con\_conj, \{\neg c\}) \equiv \{a \vee b, c\} \cup \{c \wedge \neg a\} \vdash view_2$
- Cooperativeness is not ensured for db'_1:
  $\text{neg}(view_1, \{\neg c\}) \cup \{\neg c\} \cup con \equiv \{a \vee b, \neg c\} \cup \{\neg c \Rightarrow a\} \not\vdash b \in view'_2$

Definition (Local Optimality)

A proper truthful protocol P is said to be *locally optimal*, if for each proper truthful protocol $\tilde{P}$ on every input such that

- the input is admissible
- its processing not necessarily ends up in an insecure state

it holds that

- (Least Failed Updates) $\tilde{P}$ performs strictly less updates than P,

or

- (Most Informative) $\tilde{P}$ performs the same updates as P, but offers at most the information provided by P (i.e., $view_i^P \vdash view_i^{\tilde{P}}$).

Theorem (Local Optimality)

The proposed view-update transaction protocol is locally optimal.

Local Optimality

Sketch of Proof:

We must show:

- The proposed protocol (Protocol 2) is proper truthful deterministic protocol.
- For any admissible input, any proper truthful deterministic protocol P:
  - has more failed updates than Protocol 2 **or**
  - is at most as informative as Protocol 2.

We outline the proof for the situation of a *potential conflict between integrity and confidentiality*, i.e.,

- Integrity violation possible:

$$\mathsf{neg}(view_{i-1}, Inc\Delta) \not\models con$$

- Notification of integrity violation discloses secret:

$$view_{i-1} \cup \mathsf{neg}(\neg con\_conj, Inc\Delta) \vdash \mathsf{ccp}(\psi, S\Delta_{i-1}) \qquad \text{for a } \psi \in psec(CCP)$$

or

$$view_{i-1} \cup \mathsf{neg}(\neg con\_conj, Inc\Delta) \vdash \psi \qquad \text{for a } \psi \in psec(TCP)$$

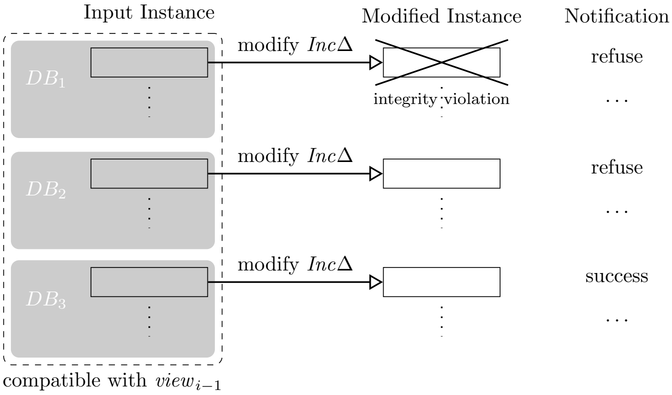
- P: proper truthful deterministic view-update transaction protocol.
- The client may simulate P on each admissible db'_{i-1}, i.e., $db'_{i-1} \models view_{i-1}$.
- In simulating, the client may distinguish three sets of admissible instances:
  - **Refusal due to immediate integrity-confidentiality conflict**: failing integrity check
    $DB_1 = \{db'_{i-1} \mid db'_{i-1} \models view_{i-1} \cup \mathsf{neg}(\neg con\_conj, Inc\Delta)\}$
  - **Additional refusal**: passing integrity check, but not updated
    $DB_2 = \{db'_{i-1} \mid db'_{i-1} \models view_{i-1} \cup \mathsf{neg}(con, Inc\Delta)$ and $P(con, db'_{i-1}, \ldots) = (., db'_{i-1}, \ldots)\}$
  - **No refusal**: passing integrity check and updated
    $DB_3 = \{db'_{i-1} \mid db'_{i-1} \models view_{i-1} \cup \mathsf{neg}(con, Inc\Delta)$ and $P(con, db'_{i-1}, \ldots) = (., db'^{Inc\Delta}_{i-1}, \ldots)\}$

We can prove the following:

$$\text{(Integrity Violation Possible)} \qquad DB_1 \neq \emptyset.$$
$$\text{(Additional Refusal Needed)} \qquad DB_2 \neq \emptyset.$$
$$\text{(Always Refused)} \qquad DB_3 = \emptyset.$$

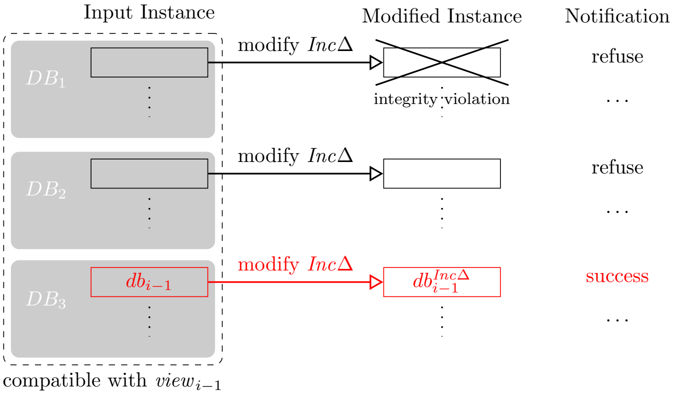
From the last point we can conclude that P does not perform an update (like Protocol 2 in the studied situation).

Sketched Proof of $DB_3 = \emptyset$ by Contradiction



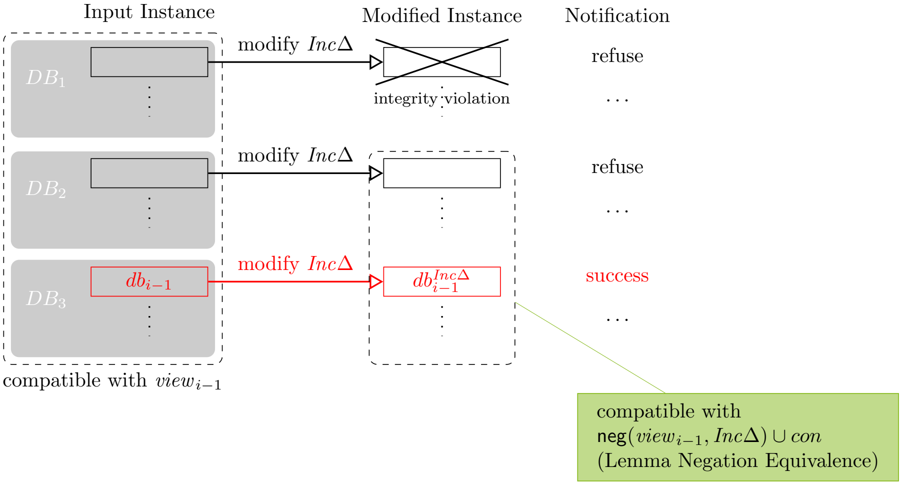
Overall situation

Sketched Proof of $DB_3 = \emptyset$ by Contradiction

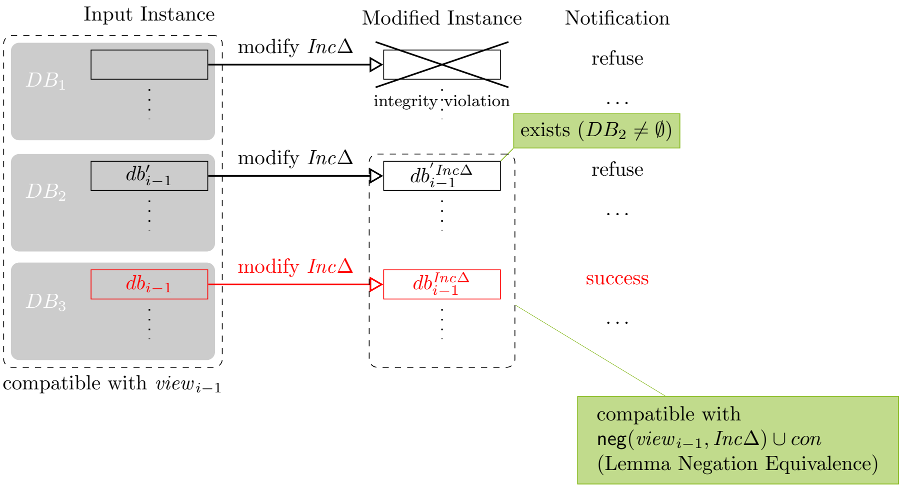


Assume current instance $db_{i-1} \in DB_3$ (No refusal)

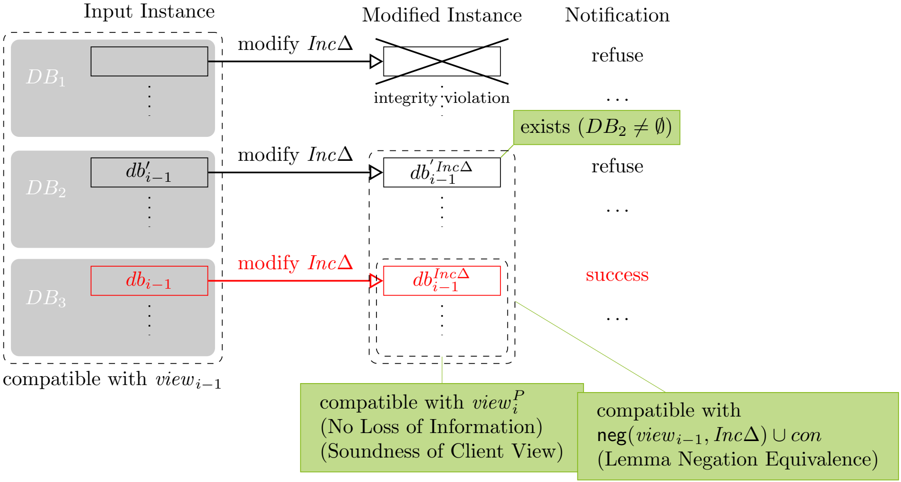
Sketched Proof of $DB_3 = \emptyset$ by Contradiction



Sketched Proof of $DB_3 = \emptyset$ by Contradiction

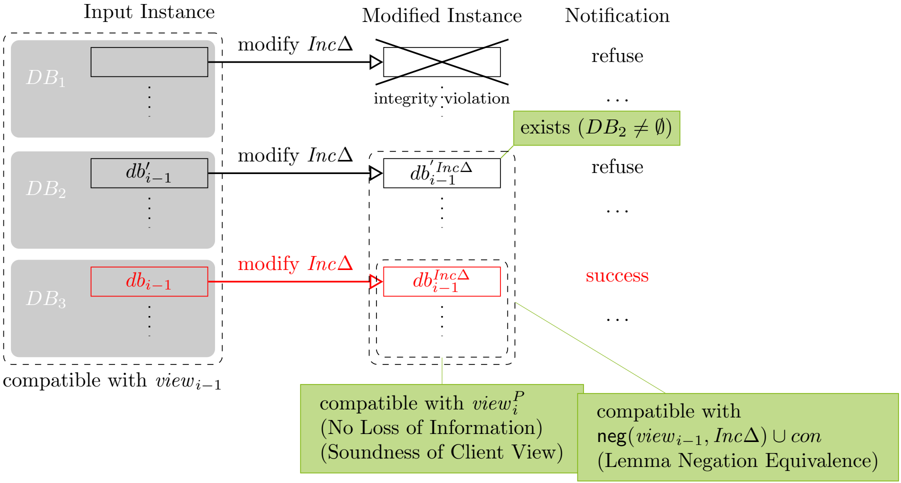


Sketched Proof of $DB_3 = \emptyset$ by Contradiction



No Loss of Information	Models of $view^P_i$ make valid neg$(view_{i-1}, Inc\Delta) \cup con$
Soundness of Client View	Client cannot distinguish model of $view^P_i$ from db_i by observed success.

Sketched Proof of $DB_3 = \emptyset$ by Contradiction



Hence, neg$(view_{i-1}, Inc\Delta) \cup con \not\models view^P_i$ (contradicts Cooperativeness) with $db'^{Inc\Delta}_{i-1}$ as witness of non-implication $\square$