Election Verifiability with ProVerif

Vincent Cheval[1], *Véronique Cortier*[2], ***Alexandre Debant***[2]

[1]Inria Paris, France
[2]Université de Lorraine, Inria, CNRS, Nancy, France

CSF 2023
Dubrovnik, Croatia

Security properties

Vote secrecy

“No one should know who I voted for”

Verifiability

“No one can modify the outcome of the election”

Security properties

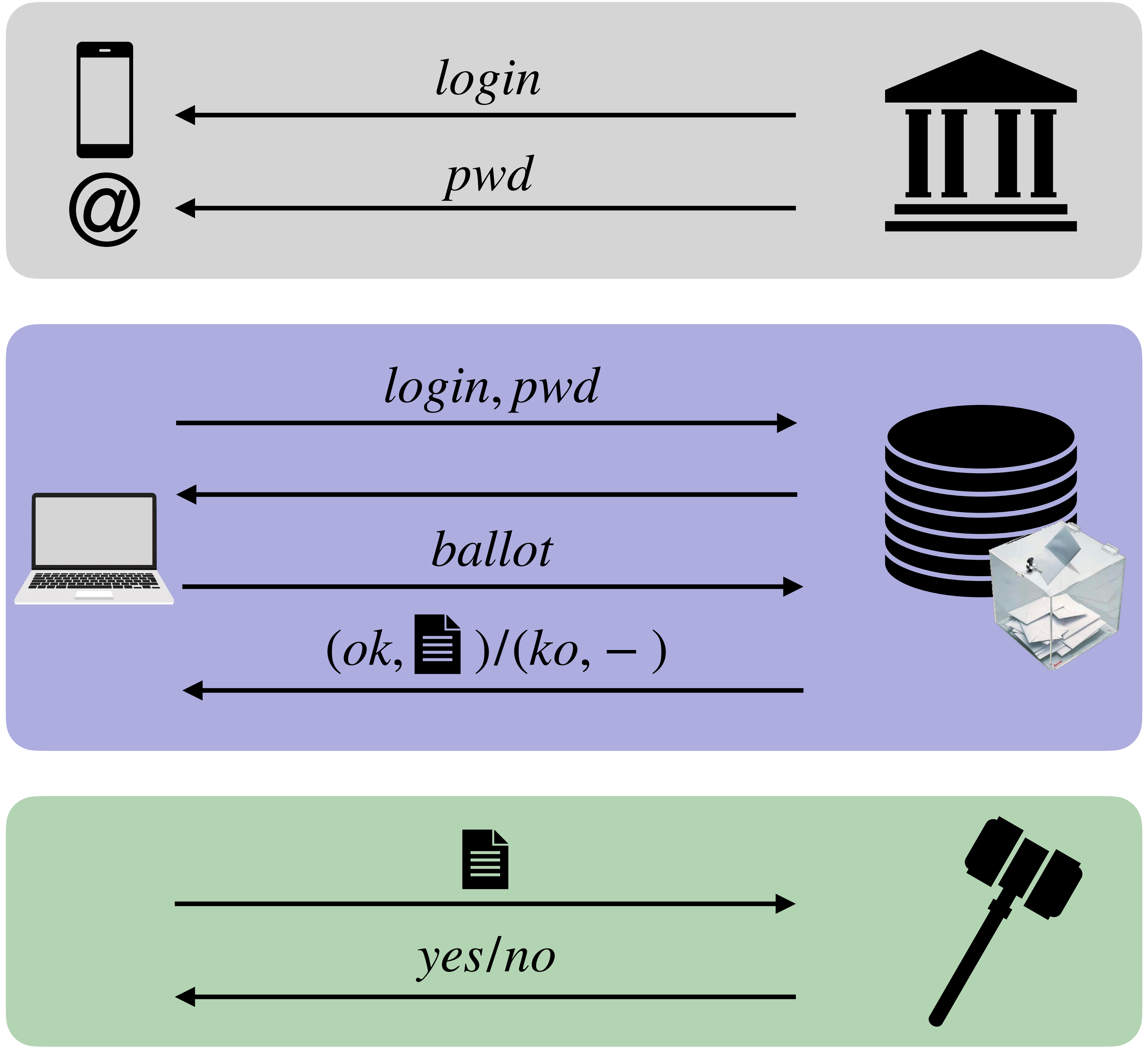
Vote secrecy

“No one should know who I voted for”

Verifiability

“No one can modify the outcome of the election”

E-voting protocol
- overview -



Setup phase

Voting phase

Verification phase

E2E verifiability

[Cortier et al - ESORICS'14]

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $V_{\mathtt{HV}}$ is the multiset of votes of honest voters who verify
- $V'_{\mathtt{HNV}}$ is a submultiset of the multiset of votes of honest voters who do not verify
- $V_{\mathtt{D}}$ contains at most one vote per dishonest voter

E2E verifiability

[Cortier et al - ESORICS'14]

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $V_{\mathtt{HV}}$ is the multiset of votes of honest voters who verify
- $V'_{\mathtt{HNV}}$ is a submultiset of the multiset of votes of honest voters who d
- $V_{\mathtt{D}}$ contains at most one vote per dishonest voter

Cannot be check directly with existing tools…

E2E verifiability

[Cortier et al - ESORICS’14]

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $V_{\mathtt{HV}}$ is the multiset of votes of honest voters who verify
- $V'_{\mathtt{HNV}}$ is a submultiset of the multiset of votes of honest voters who d
- $V_{\mathtt{D}}$ contains at most one vote per dishonest voter

Cannot be check directly
with existing tools…

Approaches based on sub-properties

e.g, [Cortier et al - CSF’19], [Baloglu et al - CSF’21]

- **Eligibility:** each vote has been cast by a legitimate voter
- **Individual** verifiability
  - **Cast-as-intended:** the voter’s ballot contains their intended vote
  - **Recorded-as-cast:** the counted ballot corresponds to the cast one
- **Universal verifiability:** the result corresponds to the content of the ballot-box
- **No clash attacks:** two voters cannot agree on the same ballot

E2E verifiability

[Cortier et al - ESORICS'14]

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $V_{\mathtt{HV}}$ is the multiset of votes of honest voters who verify
- $V'_{\mathtt{HNV}}$ is a submultiset of the multiset of votes of honest voters who d
- $V_{\mathtt{D}}$ contains at most one vote per dishonest voter

Cannot be check directly with existing tools...

Approaches based on sub-properties

e.g, [Cortier et al - CSF'19], [Baloglu et al - C

- **Eligibility:** each vote has been cast by a legitimate voter
- **Individual** verifiability
  - **Cast-as-intended:** the voter's ballot contains their intended vot
  - **Recorded-as-cast:** the counted ballot corresponds to the ca
- **Universal verifiability:** the result corresponds to the content o box
- **No clash attacks:** two voters cannot agree on the same ballo

These are only sufficient conditions...

Contributions

1. **Exact characterization of E2E verifiability**

Theorem - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

Contributions

1. **Exact characterization of E2E verifiability**

Theorem - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

2. **A ProVerif framework to analyze evoting protocols**

Applied to several protocols: Helios, Belenios, Swiss Post, CHVote

ProVerif

What is ProVerif?

- is an automatic prover for symbolic analysis
  - messages abstracted with terms
  - Dolev-Yao attacker model (intercept/inject/modify)
- can model an unbounded number of sessions
- handles trace-based properties
- handles equivalence-based properties
- has already be used to analyse voting protocols, e.g., Helios, Belenios, Swiss Post, CHVote, etc

$$
\begin{aligned}
P, Q := {} & 0 \\
& \mid \mathtt{new}\ n; P \\
& \mid \mathtt{let}\ x = v\ \mathtt{in}\ P\ \mathtt{else}\ Q; \\
& \mid \mathtt{in}(c, x); P \\
& \mid \mathtt{out}(c, u); P \\
& \mid (P \mid Q) \\
& \mid\ !P \\
& \mid \mathtt{event}\ e(u_1, \ldots, u_n); P
\end{aligned}
$$

A trace tr is a finite sequence of `in`, `out`, or $\mathtt{event}(\mathtt{e}(\mathtt{u}_1, \ldots, \mathtt{u}_\mathtt{n}))$.

Queries

Event satisfaction - A trace $tr = tr_1 \dots tr_n$ executes event $E(u_1, \dots, u_n)$ at time $\tau \in \{1,\dots,n\}$, noted $(tr, \tau) \vdash E(u_1, \dots, u_n)$, if $tr_\tau = \texttt{event}(E(u_1, \dots, u_n))$

Queries

Event satisfaction - A trace $tr = tr_1 \ldots tr_n$ executes event $E(u_1, \ldots, u_n)$ at time $\tau \in \{1,\ldots,n\}$, noted $(tr, \tau) \vdash E(u_1, \ldots, u_n)$, if $tr_\tau = \mathtt{event}(E(u_1, \ldots, u_n))$

Query formula - A trace $tr = tr_1 \ldots tr_n$ satisfies a query of the form

$$\bigwedge_{k=1}^{p} F_k(v_1, \ldots, v_{l_k}) \;\Rightarrow\; \bigvee_{i=1}^{m} \bigwedge_{j=1}^{n_i} E_{i,j}(u_1^{i,j}, \ldots, u_{l_{i,j}}^{i,j})$$

if for all substitution σ such that for all k, $(tr, \tau_k) \vdash F_k(v_1, \ldots, v_{l_k})\sigma$ for some τ_k, there exists σ' and i such that for all j, there exists $\tau_{i,j}$ such that $(tr, \tau_{i,j}) \vdash E_{i,j}(u_1^{i,j}, \ldots, u_{l_{i,j}}^{i,j})\sigma'$ and $F_k(v_1, \ldots, v_{l_k})\sigma = F_k(v_1, \ldots, v_{l_k})\sigma'$

Injective queries

Injective query - A trace $tr = tr_1 \dots tr_n$ satisfies an injective query of the form

$$\mathtt{inj}-F_0(v_0,\dots,v_{l_0}) \wedge \bigwedge_{k=1}^{p} F_k(v_1,\dots,v_{l_k}) \Rightarrow$$
$$\bigvee_{i=1}^{m} \mathtt{inj}-E_{i,0}(u_1^{i,0},\dots,u_{l_{i,0}}^{i,0}) \wedge \bigwedge_{j=1}^{n_i} E_{i,j}(u_1^{i,j},\dots,u_{l_{i,j}}^{i,j})$$

if for all substitution σ such that for all k, $(tr,\tau_k) \vdash F_k(v_1,\dots,v_{l_k})\sigma$ for some τ_k, there exists σ' and i such that for all j, there exists $\tau_{i,j}$ such that $(tr,\tau_{i,j}) \vdash E_{i,j}(u_1^{i,j},\dots,u_{l_{i,j}}^{i,j})\sigma'$ and $F_k(v_1,\dots,v_{l_k})\sigma = F_k(v_1,\dots,v_{l_k})\sigma'$.

Moreover, there exists an injective function $f : \mathcal{F}_0(tr) \rightarrow \mathcal{E}_0(tr)$ such that if $(tr,\alpha) \vdash F_0(v_1,\dots,v_{l_0})\sigma$ then $(tr,f(\alpha)) \vdash E_{i,0}(u_1^{i,0},\dots,u_{l_{i,0}}^{i,0})\sigma'$.

$\mathcal{F}_0(tr), \mathcal{E}_0(tr) \subseteq \{1,\dots,n\}$ are the sets of indices matching respectively $F_0(v_0,\dots,v_{l_0})$ and $E_{i,0}(u_1^{i,0},\dots,u_{l_{i,0}}^{i,0})$

Injective queries

Injective query - A trace $tr = tr_1 \dots tr_n$ satisfies an injective query of the form

Example: $\rho = \mathtt{inj} - F_0(x) \Rightarrow \mathtt{inj} - E_0(x)$

$\quad\quad \vee\ \mathtt{inj} - E_1(x)$

if at

fo

M

$(t$

$\mathscr{F}$

Injective queries

Injective query - A trace $tr = tr_1 \dots tr_n$ satisfies an injective query of the form

Example: $\rho = \mathtt{inj}-F_0(x) \Rightarrow \mathtt{inj}-E_0(x)$

$\qquad\qquad\qquad\qquad\qquad\qquad \vee\ \mathtt{inj}-E_1(x)$

- $tr_1 = \mathtt{event}(E_0(a))\,.\,\mathtt{event}(E_1(a))\,.\,\mathtt{event}(F_0(a))\,.\,\mathtt{event}(F_0(a))$

if … at

fo

M

$(t$

$\mathscr{F}$

Injective queries

Injective query - A trace $tr = tr_1 \dots tr_n$ satisfies an injective query of the form

if

fo

M

$(t$

$\mathcal{F}$

at

Example: $\rho = \mathtt{inj}-F_0(x) \Rightarrow \mathtt{inj}-E_0(x)$
$\vee\ \mathtt{inj}-E_1(x)$

- $tr_1 = \mathtt{event}(E_0(a))\,.\,\mathtt{event}(E_1(a))\,.\,\mathtt{event}(F_0(a))\,.\,\mathtt{event}(F_0(a))$

f

tr_1 satisfies ρ ✓

Injective queries

Injective query - A trace $tr = tr_1 \ldots tr_n$ satisfies an injective query of the form

if

fo

M

(t

$\mathscr{F}$

at

Example: $\rho = \mathtt{inj} - F_0(x) \Rightarrow \mathtt{inj} - E_0(x)$
$\qquad\qquad\qquad\qquad \vee\ \mathtt{inj} - E_1(x)$

- $$\begin{tikzcd}
tr_1 = \mathtt{event}(E_0(a)) & .\,\mathtt{event}(E_1(a)) & .\,\mathtt{event}(F_0(a)) \arrow[ll, bend left, "f"] & .\,\mathtt{event}(F_0(a)) \arrow[ll, bend left]
\end{tikzcd}$$

$\boxed{tr_1 \text{ satisfies } \rho \ ✓}$

- $tr_2 = \mathtt{event}(E_0(a))\,.\,\mathtt{event}(F_0(a))\,.\,\mathtt{event}(F_0(a))$

Injective queries

Injective query - A trace $tr = tr_1 \ldots tr_n$ satisfies an injective query of the form

if

fo

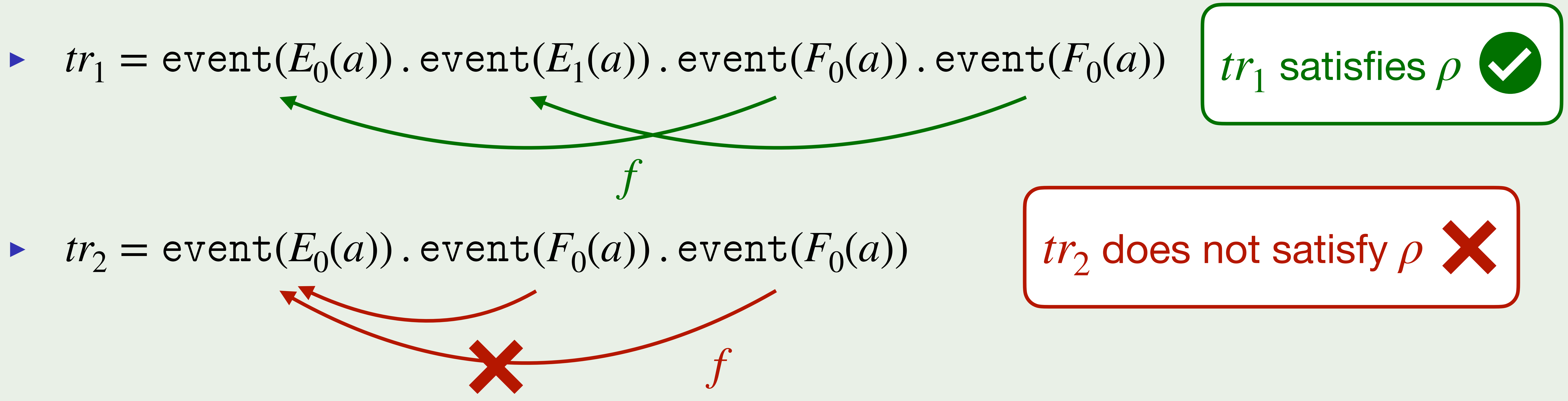
M

(t

$\mathscr{F}$

at

Example: $\rho = \texttt{inj} - F_0(x) \Rightarrow \texttt{inj} - E_0(x)$
$\vee\ \texttt{inj} - E_1(x)$



- $tr_1 = \texttt{event}(E_0(a))\,.\,\texttt{event}(E_1(a))\,.\,\texttt{event}(F_0(a))\,.\,\texttt{event}(F_0(a))$

tr_1 satisfies ρ ✓

- $tr_2 = \texttt{event}(E_0(a))\,.\,\texttt{event}(F_0(a))\,.\,\texttt{event}(F_0(a))$

tr_2 does not satisfy ρ ✗

E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:

- $\texttt{hv}(\texttt{id})$, an honest voter who verifies
- $\texttt{hnv}(\texttt{id})$, an honest voter who does not verify
- $\texttt{corrupt}(\texttt{id})$, a dishonest voter

Protocol steps

- $\texttt{voted}(id, v)$, voter id has cast a vote v
- $\texttt{verified}(id, v)$, voter id has cast a vote v and verified
- $\texttt{counted}(v)$, a vote for v has been counted during the tally
- $\texttt{finish}$, the tally has been completed

E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:

- $\mathtt{hv}(\mathtt{id})$, an honest voter who verifies
- $\mathtt{hnv}(\mathtt{id})$, an honest voter who does not verify
- $\mathtt{corrupt}(\mathtt{id})$, a dishonest voter

Protocol steps

- $\mathtt{voted}(id, v)$, voter id has cast a vote v
- $\mathtt{verified}(id, v)$, voter id has cast a vote v and verified
- $\mathtt{counted}(v)$, a vote for v has been counted during the tally
- $\mathtt{finish}$, the tally has been completed

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $result = \{\!\{ v \mid (tr, \tau) \vdash \mathtt{counted(v)} \}\!\}$
- $V_{\mathtt{HV}} = \{\!\{ v \mid (tr, \tau) \vdash \mathtt{verified}(id, v)$ and $(tr, \tau') \vdash \mathtt{hv}(id) \}\!\}$
- $V'_{\mathtt{HNV}} \subseteq_m V_{\mathtt{HNV}} = \{\!\{ v \mid (tr, \tau) \vdash \mathtt{voted}(id, v)$ and $(tr, \tau') \vdash \mathtt{hnv}(id) \}\!\}$
- $|V_{\mathtt{D}}| \leq |\mathtt{D}|$ where $D = \{ id \mid (tr, \tau) \vdash \mathtt{corrupt}(id) \}$

E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:

- $\texttt{hv}(\texttt{id})$, an honest voter who verifies
- $\texttt{hnv}(\texttt{id})$, an honest voter who does not verify
- $\texttt{corrupt}(\texttt{id})$, a dishonest voter

Protocol steps

- $\texttt{voted}(id, v)$, voter id has cast a vote v
- $\texttt{verified}(id, v)$, voter id has cast a vote v and verified
- $\texttt{counted}(v)$, a vote for v has been counted during the tally
- $\texttt{finish}$, the tally has been completed

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\texttt{HV}} \uplus V'_{\texttt{HNV}} \uplus V_{\texttt{D}}$$

where:

- $result = \{\!\!\{ v \mid (tr, \tau) \vdash \texttt{counted(v)} \}\!\!\}$

all the counted votes

- $V_{\texttt{HV}} = \{\!\!\{ v \mid (tr, \tau) \vdash \texttt{verified}(id, v) \text{ and } (tr, \tau') \vdash \texttt{hv}(id) \}\!\!\}$
- $V'_{\texttt{HNV}} \subseteq_m V_{\texttt{HNV}} = \{\!\!\{ v \mid (tr, \tau) \vdash \texttt{voted}(id, v) \text{ and } (tr, \tau') \vdash \texttt{hnv}(id) \}\!\!\}$
- $|V_{\texttt{D}}| \leq |\texttt{D}|$ where $D = \{ id \mid (tr, \tau) \vdash \texttt{corrupt}(id) \}$

E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:

- $\mathtt{hv}(\mathtt{id})$, an honest voter who verifies
- $\mathtt{hnv}(\mathtt{id})$, an honest voter who does not verify
- $\mathtt{corrupt}(\mathtt{id})$, a dishonest voter

Protocol steps

- $\mathtt{voted}(id, v)$, voter id has cast a vote v
- $\mathtt{verified}(id, v)$, voter id has cast a vote v and verified
- $\mathtt{counted}(v)$, a vote for v has been counted during the tally
- $\mathtt{finish}$, the tally has been completed

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $result = \{\!\{ v \mid (tr, \tau) \vdash \mathtt{counted}(\mathtt{v}) \}\!\}$
- $V_{\mathtt{HV}} = \{\!\{ v \mid (tr, \tau) \vdash \mathtt{verified}(id, v) \text{ and } (tr, \tau') \vdash \mathtt{hv}(id) \}\!\}$
- $V'_{\mathtt{HNV}} \subseteq_m V_{\mathtt{HNV}} = \{\!\{ v \mid (tr, \tau) \vdash \mathtt{voted}(id, v) \text{ and } (tr, \tau') \vdash \mathtt{hnv}(id) \}\!\}$
- $|V_{\mathtt{D}}| \leq |\mathtt{D}|$ where $D = \{ id \mid (tr, \tau) \vdash \mathtt{corrupt}(id) \}$

all the counted votes

all the votes of honest voters who verify

E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:

- $\mathtt{hv}(\mathtt{id})$, an honest voter who verifies
- $\mathtt{hnv}(\mathtt{id})$, an honest voter who does not verify
- $\mathtt{corrupt}(\mathtt{id})$, a dishonest voter

Protocol steps

- $\mathtt{voted}(id, v)$, voter id has cast a vote v
- $\mathtt{verified}(id, v)$, voter id has cast a vote v and verified
- $\mathtt{counted}(v)$, a vote for v has been counted during the tally
- $\mathtt{finish}$, the tally has been completed

Definition - **An evoting protocol satisfies E2E verifiability if for any execution,**

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $result = \{\!\!\{ v \mid (tr, \tau) \vdash \mathtt{counted(v)} \}\!\!\}$
- $V_{\mathtt{HV}} = \{\!\!\{ v \mid (tr, \tau) \vdash \mathtt{verified}(id, v) \text{ and } (tr, \tau') \vdash \mathtt{hv}(id) \}\!\!\}$
- $V'_{\mathtt{HNV}} \subseteq_m V_{\mathtt{HNV}} = \{\!\!\{ v \mid (tr, \tau) \vdash \mathtt{voted}(id, v) \text{ and } (tr, \tau') \vdash \mathtt{hnv}(id) \}\!\!\}$
- $|V_{\mathtt{D}}| \leq |\mathtt{D}|$ where $D = \{ id \mid (tr, \tau) \vdash \mathtt{corrupt}(id) \}$

all the counted votes

all the votes of honest voters who verify

A subset of the votes of honest voters who do not verify

E2E verifiability

Events used to model E2E verifiability

Honesty and behavior of voter:

- $\mathtt{hv}(\mathtt{id})$, an honest voter who verifies
- $\mathtt{hnv}(\mathtt{id})$, an honest voter who does not verify
- $\mathtt{corrupt}(\mathtt{id})$, a dishonest voter

Protocol steps

- $\mathtt{voted}(id, v)$, voter id has cast a vote v
- $\mathtt{verified}(id, v)$, voter id has cast a vote v and verified
- $\mathtt{counted}(v)$, a vote for v has been counted during the tally
- $\mathtt{finish}$, the tally has been completed

Definition - An evoting protocol satisfies E2E verifiability if for any execution,

$$\text{result} = V_{\mathtt{HV}} \uplus V'_{\mathtt{HNV}} \uplus V_{\mathtt{D}}$$

where:

- $result = \{\!\!\{ v \mid (tr, \tau) \vdash \mathtt{counted(v)} \}\!\!\}$
- $V_{\mathtt{HV}} = \{\!\!\{ v \mid (tr, \tau) \vdash \mathtt{verified}(id, v) \text{ and } (tr, \tau') \vdash \mathtt{hv}(id) \}\!\!\}$
- $V'_{\mathtt{HNV}} \subseteq_m V_{\mathtt{HNV}} = \{\!\!\{ v \mid (tr, \tau) \vdash \mathtt{voted}(id, v) \text{ and } (tr, \tau') \vdash \mathtt{hnv}(id) \}\!\!\}$
- $|V_{\mathtt{D}}| \leq |\mathtt{D}|$ where $D = \{ id \mid (tr, \tau) \vdash \mathtt{corrupt}(id) \}$

all the counted votes

all the votes of honest voters who verify

A subset of the votes of honest voters who do not verify

At most 1 vote per dishonest voter

Exact characterization of E2E verifiability

Theorem - An evoting protocol satisfies E2E verifiability **if and only if** it all its traces tr satisfy:

- (Query 1) $\texttt{finish} \wedge \texttt{inj} - \texttt{counted}(x) \Rightarrow \texttt{inj} - \texttt{hv}(\texttt{z}) \wedge \texttt{verified}(z, x)$
  $\vee\ \texttt{inj} - \texttt{hnv}(\texttt{z}) \wedge \texttt{voted}(z, x)$
  $\vee\ \texttt{inj} - \texttt{corrupt}(\texttt{z})$
- (Query 2) $\texttt{finish} \wedge \texttt{inj} - \texttt{verified}(z, x) \Rightarrow \texttt{inj} - \texttt{counted}(\texttt{x})$

Exact characterization of E2E verifiability

Theorem - An evoting protocol satisfies E2E verifiability if and only if it all its traces tr satisfy:

- (Query 1) $\texttt{finish} \wedge \texttt{inj} - \texttt{counted}(x) \Rightarrow \texttt{inj} - \texttt{hv}(\texttt{z}) \wedge \texttt{verified}(z, x)$

  $\vee\ \texttt{inj} - \texttt{hnv}(\texttt{z}) \wedge \texttt{voted}(z, x)$

  $\vee\ \texttt{inj} - \texttt{corrupt}(\texttt{z})$

- (Query 2) $\texttt{finish} \wedge \texttt{inj} - \texttt{verified}(z, x) \Rightarrow \texttt{inj} - \texttt{counted}(\texttt{x})$

Ideas of the proof

$\boxed{\Rightarrow}$ “easy”, we can straightforwardly verify the queries

Exact characterization of E2E verifiability

Theorem - An evoting protocol satisfies E2E verifiability if and only if it all its traces tr satisfy:

- (Query 1) $$\begin{aligned} \mathtt{finish} \wedge \mathtt{inj}-\mathtt{counted}(x) \Rightarrow\ & \mathtt{inj}-\mathtt{hv}(\mathtt{z}) \wedge \mathtt{verified}(z,x) \\ \vee\ & \mathtt{inj}-\mathtt{hnv}(\mathtt{z}) \wedge \mathtt{voted}(z,x) \\ \vee\ & \mathtt{inj}-\mathtt{corrupt}(\mathtt{z}) \end{aligned}$$
- (Query 2) $\mathtt{finish} \wedge \mathtt{inj}-\mathtt{verified}(z,x) \Rightarrow \mathtt{inj}-\mathtt{counted}(\mathtt{x})$

Ideas of the proof

$\boxed{\Rightarrow}$ “easy”, we can straightforwardly verify the queries

$\boxed{\Leftarrow}$ “more difficult”…

$\boxed{\Leftarrow}$ idea of the proof

Assumptions - for all traces tr, Query 1 and Query 2 are satisfied.

- (Query 1) $\texttt{finish} \wedge \texttt{inj} - \texttt{counted}(x) \Rightarrow \texttt{inj} - \texttt{hv(z)} \wedge \texttt{verified}(z, x)$

  $\qquad\qquad\qquad\qquad\vee\ \texttt{inj} - \texttt{hnv(z)} \wedge \texttt{voted}(z, x)$

  $\qquad\qquad\qquad\qquad\vee\ \texttt{inj} - \texttt{corrupt(z)}$

- (Query 2) $\texttt{finish} \wedge \texttt{inj} - \texttt{verified}(z, x) \Rightarrow \texttt{inj} - \texttt{counted(x)}$

$\boxed{\Leftarrow}$ idea of the proof

Assumptions - **for all traces tr, Query 1 and Query 2 are satisfied.**

- (Query 1) $\mathtt{finish} \land \mathtt{inj}-\mathtt{counted}(x) \Rightarrow \mathtt{inj}-\mathtt{hv}(\mathtt{z}) \land \mathtt{verified}(z,x)$
  $\lor\ \mathtt{inj}-\mathtt{hnv}(\mathtt{z}) \land \mathtt{voted}(z,x)$
  $\lor\ \mathtt{inj}-\mathtt{corrupt}(\mathtt{z})$
- (Query 2) $\mathtt{finish} \land \mathtt{inj}-\mathtt{verified}(z,x) \Rightarrow \mathtt{inj}-\mathtt{counted}(\mathtt{x})$

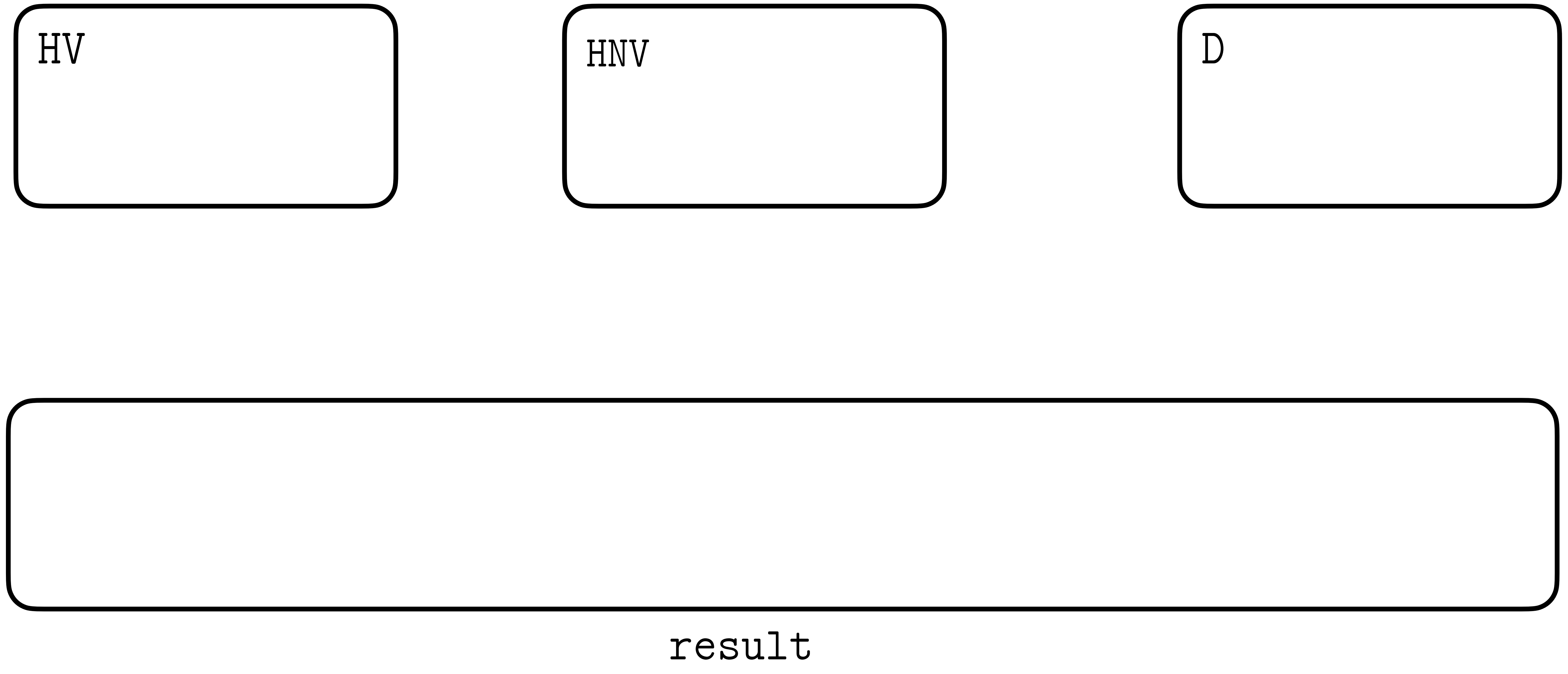
🎯 **Goal:** define an injective function $h : \mathtt{result} \rightarrow \mathtt{HV} \uplus \mathtt{HNV} \uplus \mathtt{D}$ that is surjective over $\mathtt{HV}$

⇐ idea of the proof

Assumptions - **for all traces tr, Query 1 and Query 2 are satisfied.**

- (Query 1) $$\begin{aligned}\mathtt{finish} \land \mathtt{inj}-\mathtt{counted}(x) \Rightarrow\ & \mathtt{inj}-\mathtt{hv(z)} \land \mathtt{verified}(z,x)\\ \lor\ & \mathtt{inj}-\mathtt{hnv(z)} \land \mathtt{voted}(z,x)\\ \lor\ & \mathtt{inj}-\mathtt{corrupt(z)}\end{aligned}$$
- (Query 2) $\mathtt{finish} \land \mathtt{inj}-\mathtt{verified}(z,x) \Rightarrow \mathtt{inj}-\mathtt{counted(x)}$

Goal: define an injective function $h : \mathtt{result} \rightarrow \mathrm{HV} \uplus \mathrm{HNV} \uplus \mathrm{D}$ that is surjective over HV



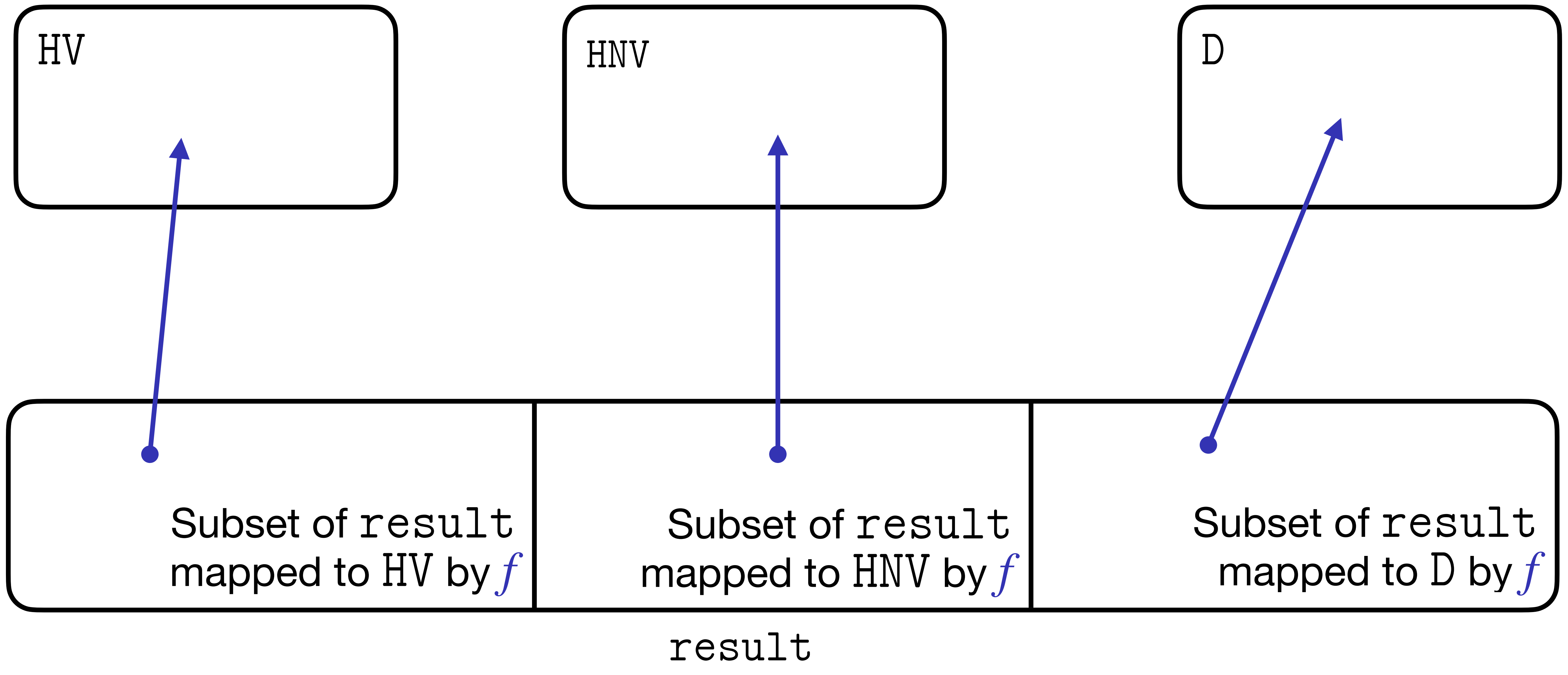
⇐ idea of the proof

Assumptions - **for all traces tr, Query 1 and Query 2 are satisfied.**

- (Query 1) $$\mathtt{finish} \wedge \mathtt{inj-counted}(x) \Rightarrow \mathtt{inj-hv(z)} \wedge \mathtt{verified}(z,x)$$ $$\vee\ \mathtt{inj-hnv(z)} \wedge \mathtt{voted}(z,x)$$ $$\vee\ \mathtt{inj-corrupt(z)}$$
- (Query 2) $$\mathtt{finish} \wedge \mathtt{inj-verified}(z,x) \Rightarrow \mathtt{inj-counted(x)}$$

Injective function f

🎯 **Goal:** define an injective function $h : \mathtt{result} \to \mathrm{HV} \uplus \mathrm{HNV} \uplus \mathrm{D}$ that is surjective over HV


HV
HNV
D
Subset of result mapped to HV by f
Subset of result mapped to HNV by f
Subset of result mapped to D by f
result

$\boxed{\Leftarrow}$ idea of the proof

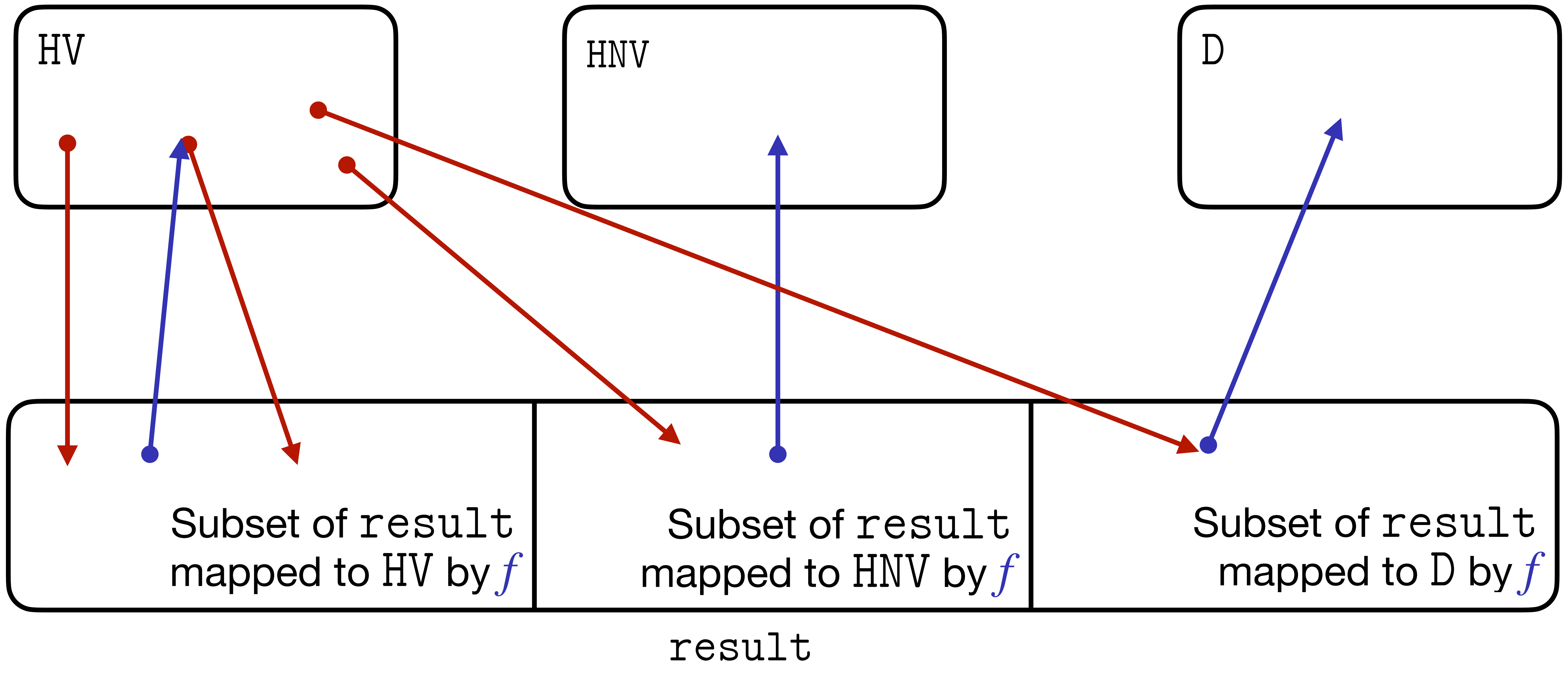
Assumptions - **for all traces tr, Query 1 and Query 2 are satisfied.**

- (Query 1) $\texttt{finish} \wedge \texttt{inj}-\texttt{counted}(x) \Rightarrow \texttt{inj}-\texttt{hv}(\texttt{z}) \wedge \texttt{verified}(z,x)$
  $\vee\ \texttt{inj}-\texttt{hnv}(\texttt{z}) \wedge \texttt{voted}(z,x)$
  $\vee\ \texttt{inj}-\texttt{corrupt}(\texttt{z})$
- (Query 2) $\texttt{finish} \wedge \texttt{inj}-\texttt{verified}(z,x) \Rightarrow \texttt{inj}-\texttt{counted}(\texttt{x})$

Injective function f

Injective function g

Goal: define an injective function $h : \texttt{result} \rightarrow \texttt{HV} \uplus \texttt{HNV} \uplus \texttt{D}$ that is surjective over $\texttt{HV}$


HV
HNV
D
Subset of result mapped to HV by f
Subset of result mapped to HNV by f
Subset of result mapped to D by f
result

idea of the proof

Assumptions - **for all traces tr, Query 1 and Query 2 are satisfied.**

- (Query 1) $$\begin{aligned}\mathtt{finish} \wedge \mathtt{inj-counted}(x) \Rightarrow\ & \mathtt{inj-hv(z)} \wedge \mathtt{verified}(z,x)\\ \vee\ & \mathtt{inj-hnv(z)} \wedge \mathtt{voted}(z,x)\\ \vee\ & \mathtt{inj-corrupt(z)}\end{aligned}$$
- (Query 2) $\mathtt{finish} \wedge \mathtt{inj-verified}(z,x) \Rightarrow \mathtt{inj-counted(x)}$

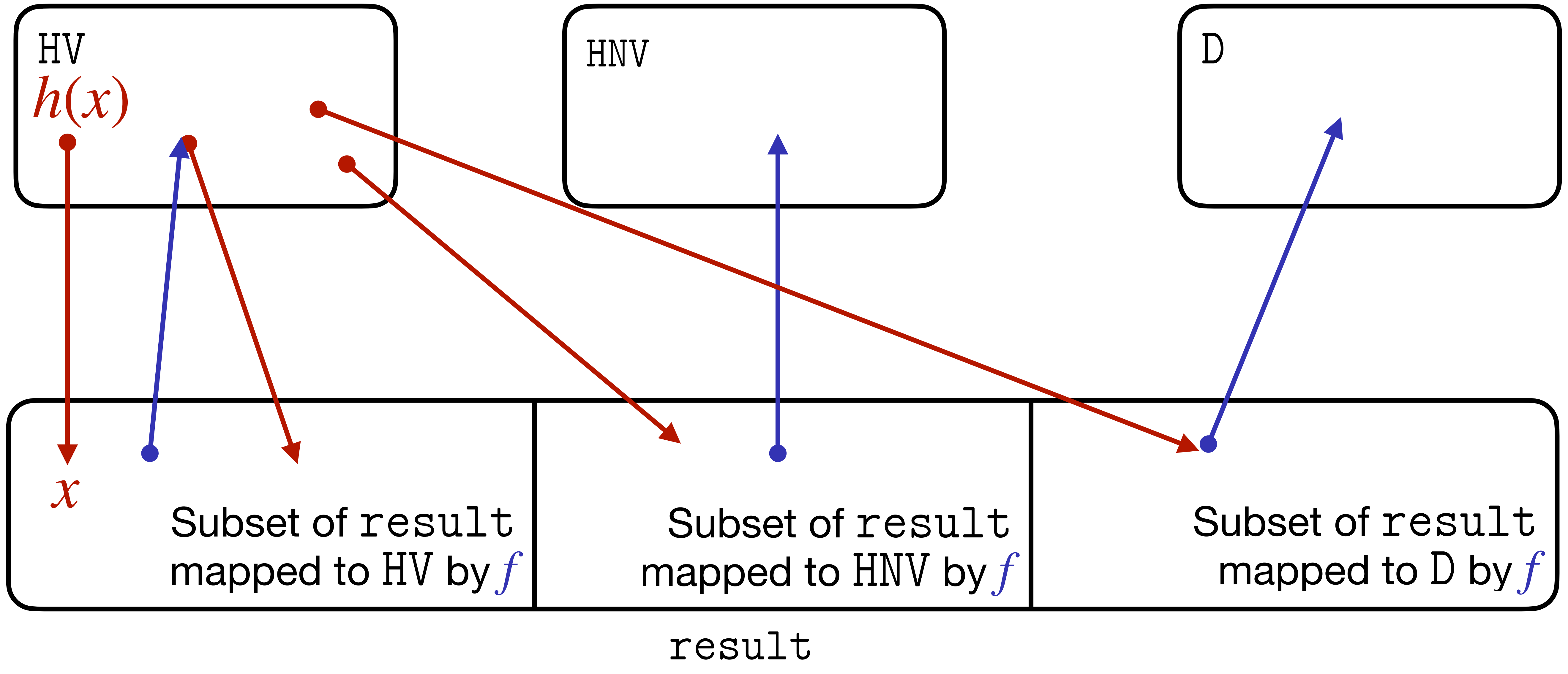
Injective function f

Injective function g

Goal: define an injective function $h : \mathtt{result} \rightarrow \mathrm{HV} \uplus \mathrm{HNV} \uplus \mathrm{D}$ that is surjective over HV

h(x) =

$$\begin{cases} g^{-1}(x) & \text{if } x \in g(\mathrm{HV}) \\ & \\ & \end{cases}$$



HV
$h(x)$
HNV
D
x
Subset of result mapped to HV by f
Subset of result mapped to HNV by f
Subset of result mapped to D by f
result

⇐ idea of the proof

Assumptions - for all traces tr, Query 1 and Query 2 are satisfied.

- (Query 1) $\texttt{finish} \wedge \texttt{inj-counted}(x) \Rightarrow \texttt{inj-hv(z)} \wedge \texttt{verified}(z,x)$
  $\vee\ \texttt{inj-hnv(z)} \wedge \texttt{voted}(z,x)$
  $\vee\ \texttt{inj-corrupt(z)}$

Injective function f

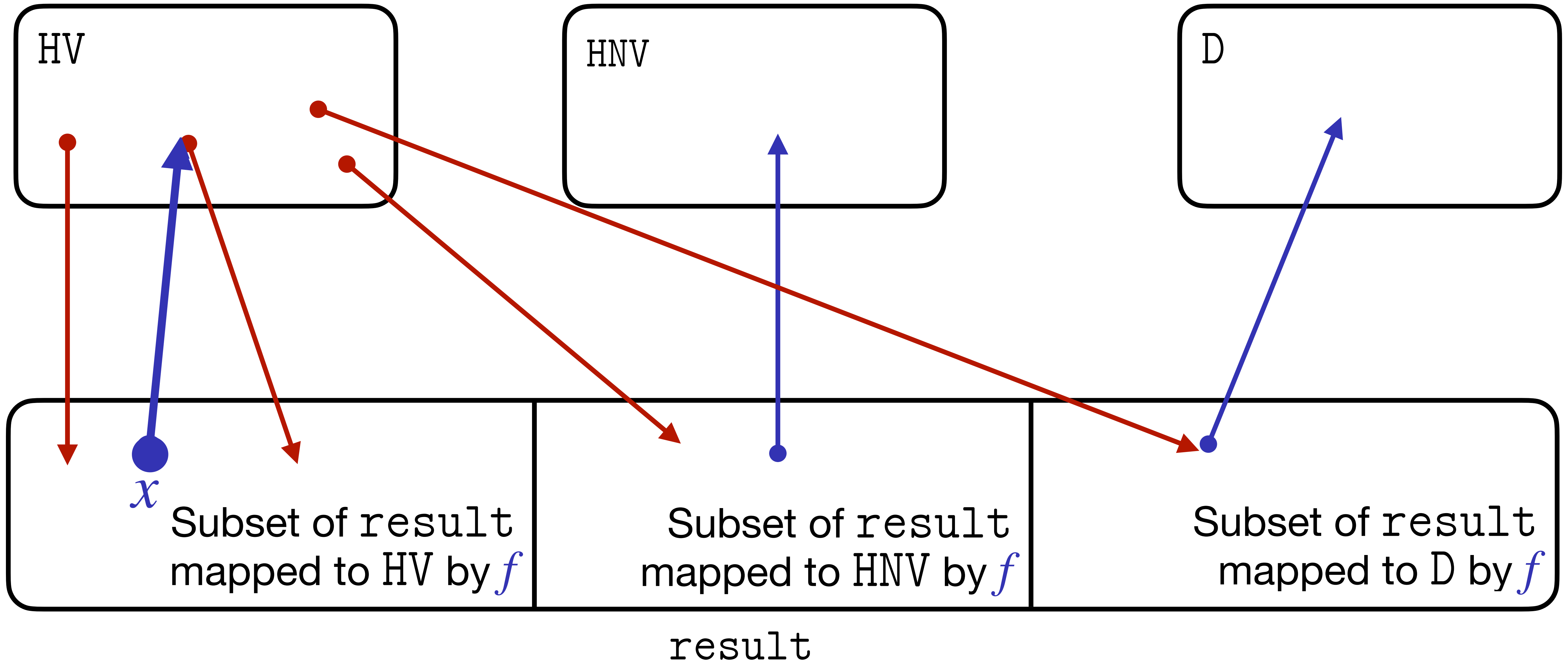
- (Query 2) $\texttt{finish} \wedge \texttt{inj-verified}(z,x) \Rightarrow \texttt{inj-counted(x)}$

Injective function g

Goal: define an injective function $h : \texttt{result} \rightarrow \texttt{HV} \uplus \texttt{HNV} \uplus \texttt{D}$ that is surjective over $\texttt{HV}$

h(x) =

$$\begin{cases} g^{-1}(x) & \text{if } x \in g(\texttt{HV}) \\ (f \circ g)^n \circ f(x) & \text{if } x \notin g(\texttt{HV}) \text{ and } f(x) \in \texttt{HV} \\ & \text{where } n = min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \texttt{HV}\} \end{cases}$$


HV
HNV
D
x
Subset of result mapped to HV by f
Subset of result mapped to HNV by f
Subset of result mapped to D by f
result

$\Leftarrow$ idea of the proof

Assumptions - for all traces tr, Query 1 and Query 2 are satisfied.

- (Query 1) $\texttt{finish} \wedge \texttt{inj}-\texttt{counted}(x) \Rightarrow \texttt{inj}-\texttt{hv(z)} \wedge \texttt{verified}(z,x)$
  $\vee\ \texttt{inj}-\texttt{hnv(z)} \wedge \texttt{voted}(z,x)$
  $\vee\ \texttt{inj}-\texttt{corrupt(z)}$
- (Query 2) $\texttt{finish} \wedge \texttt{inj}-\texttt{verified}(z,x) \Rightarrow \texttt{inj}-\texttt{counted(x)}$

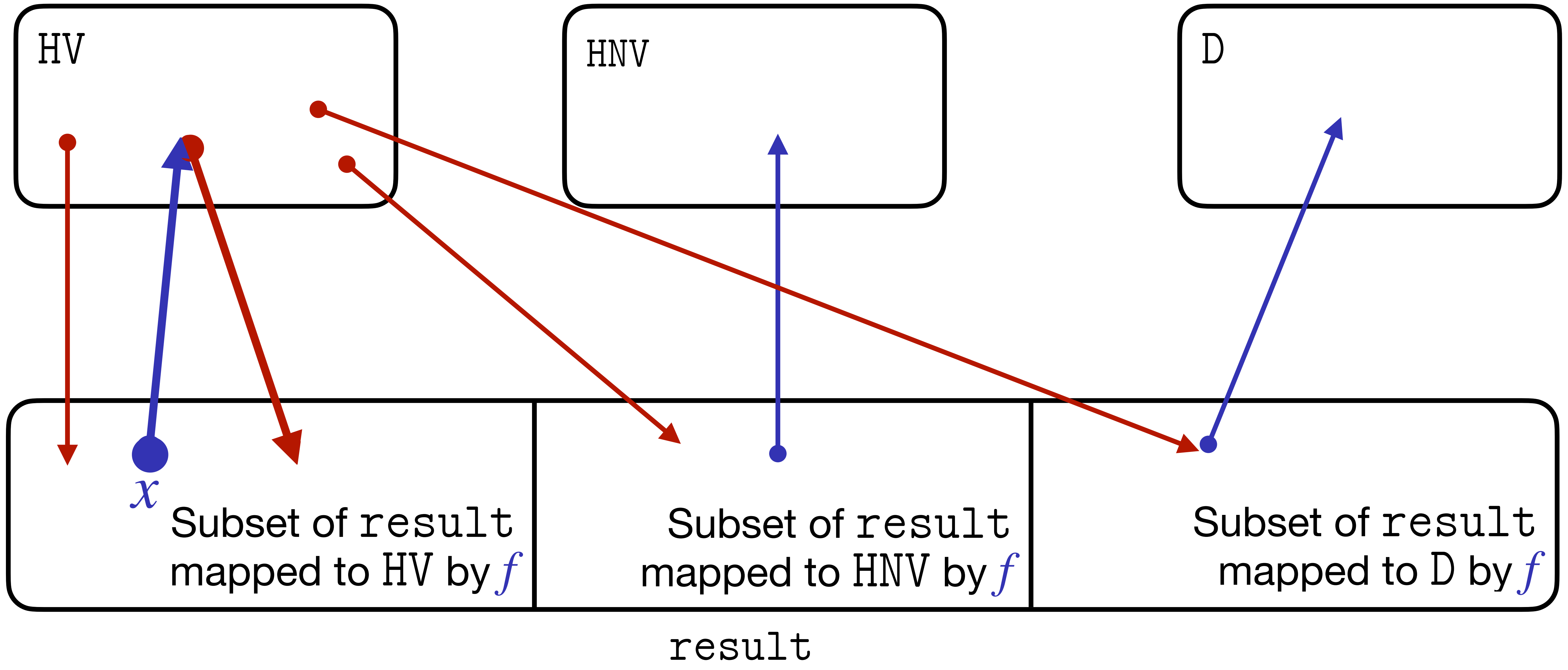
Injective function f

Injective function g

Goal: define an injective function $h : \texttt{result} \rightarrow \texttt{HV} \uplus \texttt{HNV} \uplus \texttt{D}$ that is surjective over $\texttt{HV}$

h(x) =

$$\begin{cases} g^{-1}(x) & \text{if } x \in g(\texttt{HV}) \\ (f \circ g)^n \circ f(x) & \text{if } x \notin g(\texttt{HV}) \text{ and } f(x) \in \texttt{HV} \\ & \text{where } n = min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \texttt{HV}\} \end{cases}$$


HV
HNV
D
x
Subset of result mapped to HV by f
Subset of result mapped to HNV by f
Subset of result mapped to D by f
result

⇐ idea of the proof

Assumptions - for all traces tr, Query 1 and Query 2 are satisfied.

- (Query 1) $\mathtt{finish} \wedge \mathtt{inj-counted}(x) \Rightarrow \mathtt{inj-hv(z)} \wedge \mathtt{verified}(z,x)$
  $\vee\ \mathtt{inj-hnv(z)} \wedge \mathtt{voted}(z,x)$
  $\vee\ \mathtt{inj-corrupt(z)}$
- (Query 2) $\mathtt{finish} \wedge \mathtt{inj-verified}(z,x) \Rightarrow \mathtt{inj-counted(x)}$

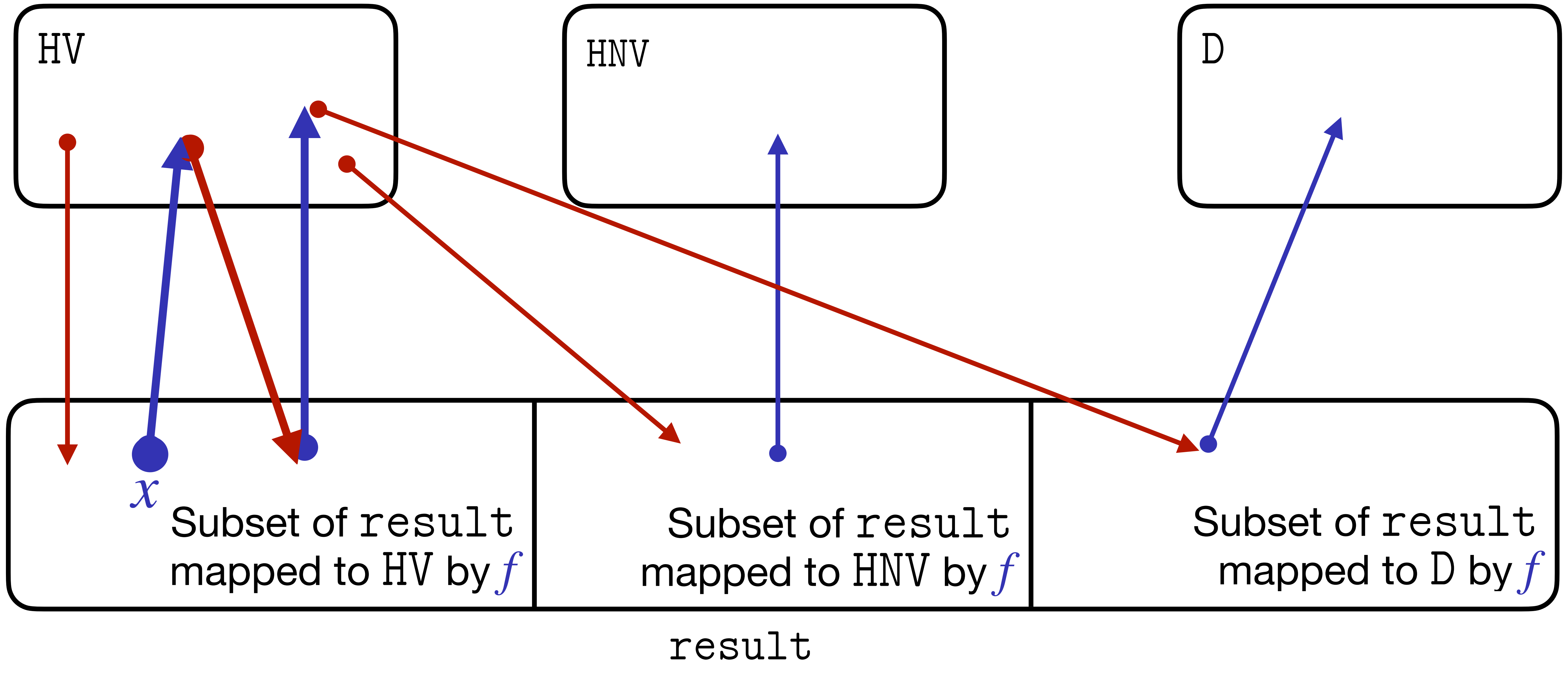
Injective function f

Injective function g

Goal: define an injective function $h : \mathtt{result} \rightarrow \mathtt{HV} \uplus \mathtt{HNV} \uplus \mathtt{D}$ that is surjective over $\mathtt{HV}$

h(x) =

$$\begin{cases} g^{-1}(x) & \text{if } x \in g(\mathtt{HV}) \\ (f\circ g)^n \circ f(x) & \text{if } x \notin g(\mathtt{HV}) \text{ and } f(x) \in \mathtt{HV} \\ & \text{where } n = min\{i > 0 \mid (f\circ g)^i \circ f(x) \notin \mathtt{HV}\} \end{cases}$$



Subset of result mapped to HV by f

Subset of result mapped to HNV by f

Subset of result mapped to D by f

⇐ idea of the proof

Assumptions - for all traces tr, Query 1 and Query 2 are satisfied.

- (Query 1) $\mathtt{finish} \wedge \mathtt{inj-counted}(x) \Rightarrow \mathtt{inj-hv(z)} \wedge \mathtt{verified}(z,x)$
  $\vee\ \mathtt{inj-hnv(z)} \wedge \mathtt{voted}(z,x)$
  $\vee\ \mathtt{inj-corrupt(z)}$

  Injective function f

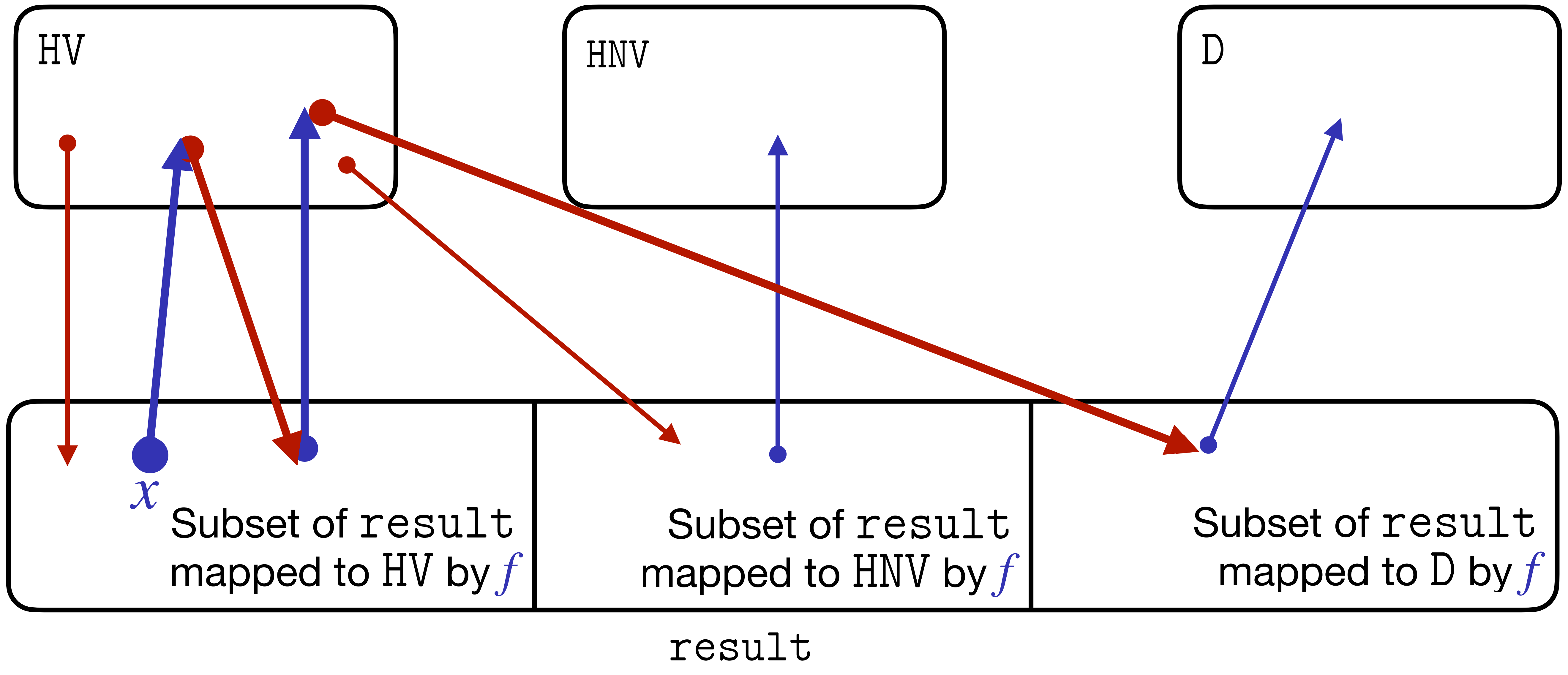
- (Query 2) $\mathtt{finish} \wedge \mathtt{inj-verified}(z,x) \Rightarrow \mathtt{inj-counted(x)}$

  Injective function g

Goal: define an injective function $h : \mathtt{result} \rightarrow \mathtt{HV} \uplus \mathtt{HNV} \uplus \mathtt{D}$ that is surjective over $\mathtt{HV}$

h(x) =

$$\begin{cases} g^{-1}(x) & \text{if } x \in g(\mathtt{HV}) \\ (f \circ g)^n \circ f(x) & \text{if } x \notin g(\mathtt{HV}) \text{ and } f(x) \in \mathtt{HV} \\ & \text{where } n = min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \mathtt{HV}\} \end{cases}$$


HV
HNV
D
x
Subset of result mapped to HV by f
Subset of result mapped to HNV by f
Subset of result mapped to D by f
result

⇐ idea of the proof

Assumptions - for all traces tr, Query 1 and Query 2 are satisfied.

- (Query 1) $\mathtt{finish} \wedge \mathtt{inj-counted}(x) \Rightarrow \mathtt{inj-hv(z)} \wedge \mathtt{verified}(z,x)$
  $\vee\ \mathtt{inj-hnv(z)} \wedge \mathtt{voted}(z,x)$
  $\vee\ \mathtt{inj-corrupt(z)}$

  Injective function f

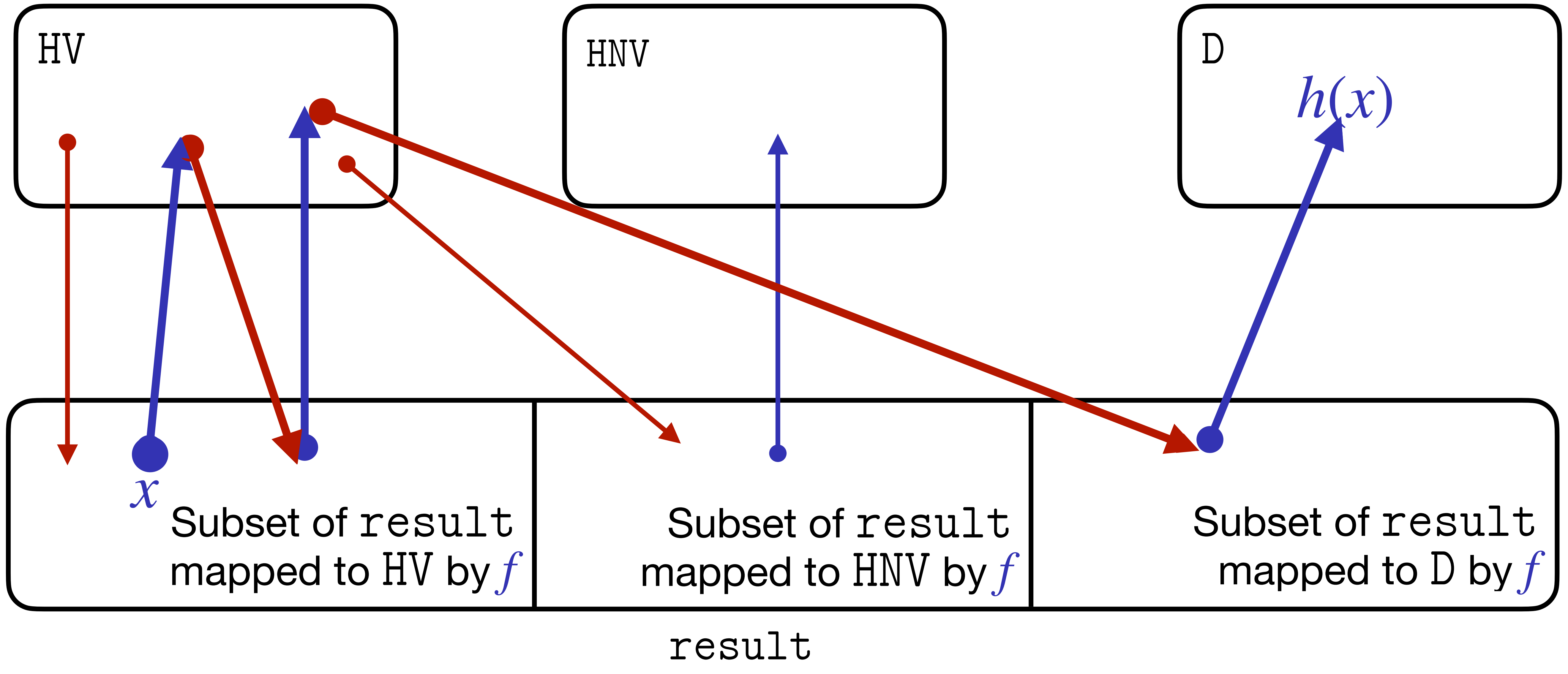
- (Query 2) $\mathtt{finish} \wedge \mathtt{inj-verified}(z,x) \Rightarrow \mathtt{inj-counted(x)}$

  Injective function g

🎯 **Goal:** define an injective function $h : \mathtt{result} \to \mathtt{HV} \uplus \mathtt{HNV} \uplus \mathtt{D}$ that is surjective over $\mathtt{HV}$

h(x) =

$$
\begin{cases}
g^{-1}(x) & \text{if } x \in g(\mathtt{HV}) \\
(f \circ g)^n \circ f(x) & \text{if } x \notin g(\mathtt{HV}) \text{ and } f(x) \in \mathtt{HV} \\
& \text{where } n = min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \mathtt{HV}\}
\end{cases}
$$


HV
HNV
D
$h(x)$
x
Subset of result mapped to HV by f
Subset of result mapped to HNV by f
Subset of result mapped to D by f
result

idea of the proof

Assumptions - for all traces tr, Query 1 and Query 2 are satisfied.

- (Query 1) $\mathtt{finish} \wedge \mathtt{inj-counted}(x) \Rightarrow \mathtt{inj-hv}(\mathtt{z}) \wedge \mathtt{verified}(z, x)$
  $\vee\ \mathtt{inj-hnv}(\mathtt{z}) \wedge \mathtt{voted}(z, x)$
  $\vee\ \mathtt{inj-corrupt}(\mathtt{z})$
- (Query 2) $\mathtt{finish} \wedge \mathtt{inj-verified}(z, x) \Rightarrow \mathtt{inj-counted}(\mathtt{x})$

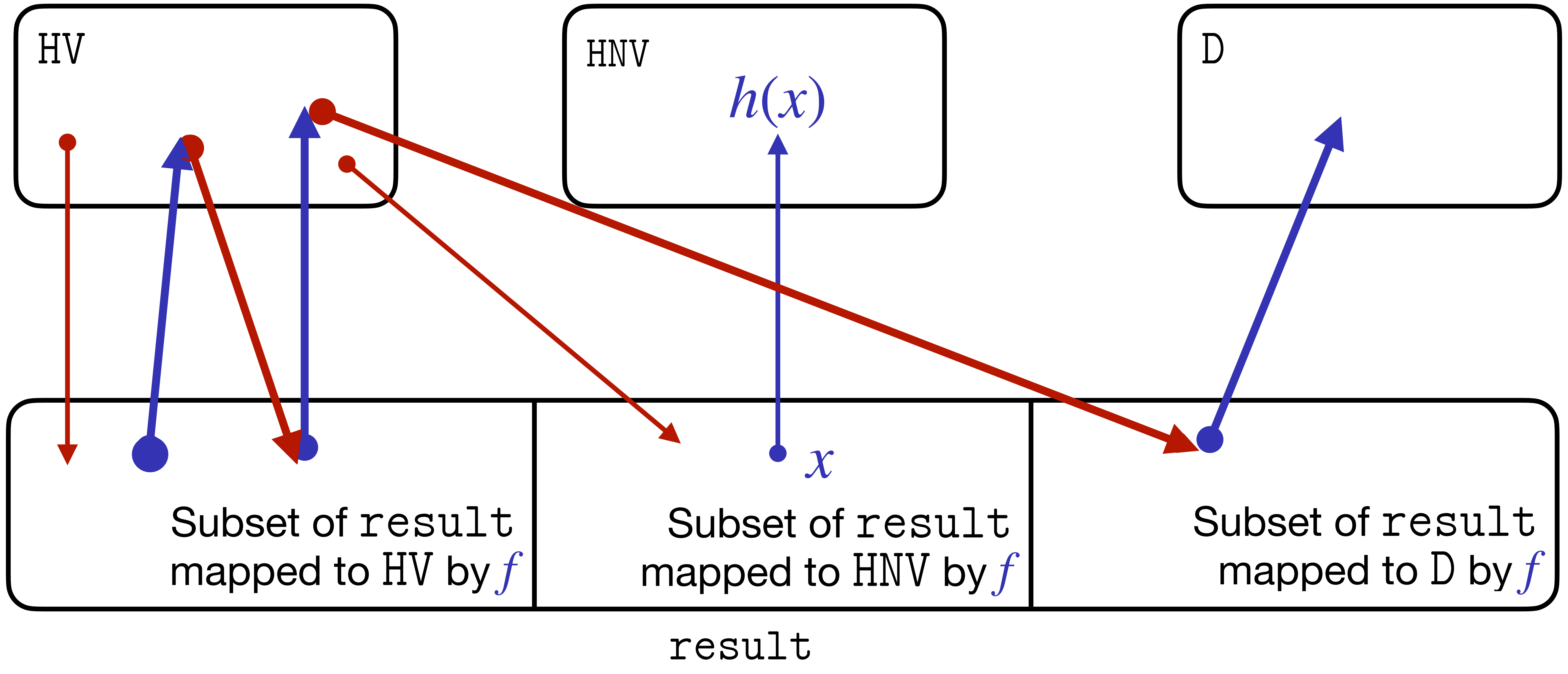
Injective function f

Injective function g

Goal: define an injective function $h : \mathtt{result} \rightarrow \mathtt{HV} \uplus \mathtt{HNV} \uplus \mathtt{D}$ that is surjective over $\mathtt{HV}$

h(x) =

$$\begin{cases} g^{-1}(x) & \text{if } x \in g(\mathtt{HV}) \\ (f \circ g)^n \circ f(x) & \text{if } x \notin g(\mathtt{HV}) \text{ and } f(x) \in \mathtt{HV} \\ & \text{where } n = min\{i > 0 \mid (f \circ g)^i \circ f(x) \notin \mathtt{HV}\} \\ f(x) & \text{otherwise} \end{cases}$$



HV

HNV

$h(x)$

D

x

Subset of $\mathtt{result}$ mapped to $\mathtt{HV}$ by f

Subset of $\mathtt{result}$ mapped to $\mathtt{HNV}$ by f

Subset of $\mathtt{result}$ mapped to $\mathtt{D}$ by f

$\mathtt{result}$

Contributions

1. **Exact characterization of E2E verifiability**

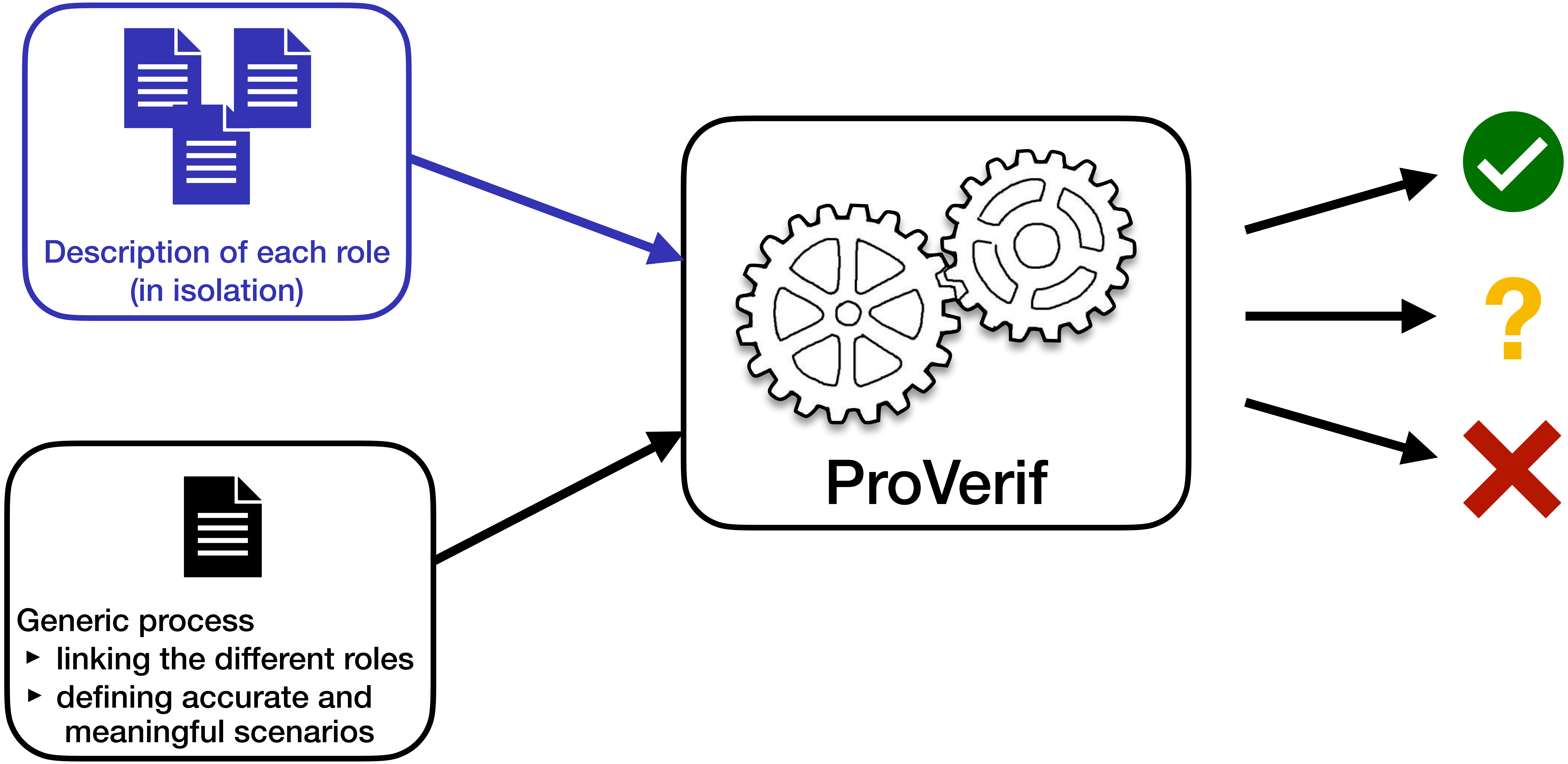
✓

Theorem - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

2. **A ProVerif framework to analyze evoting protocols**

Applied to several protocols: Helios, Belenios, Swiss Post, CHVote

Our framework



Description of each role
(in isolation)

Generic process

- linking the different roles
- defining accurate and meaningful scenarios

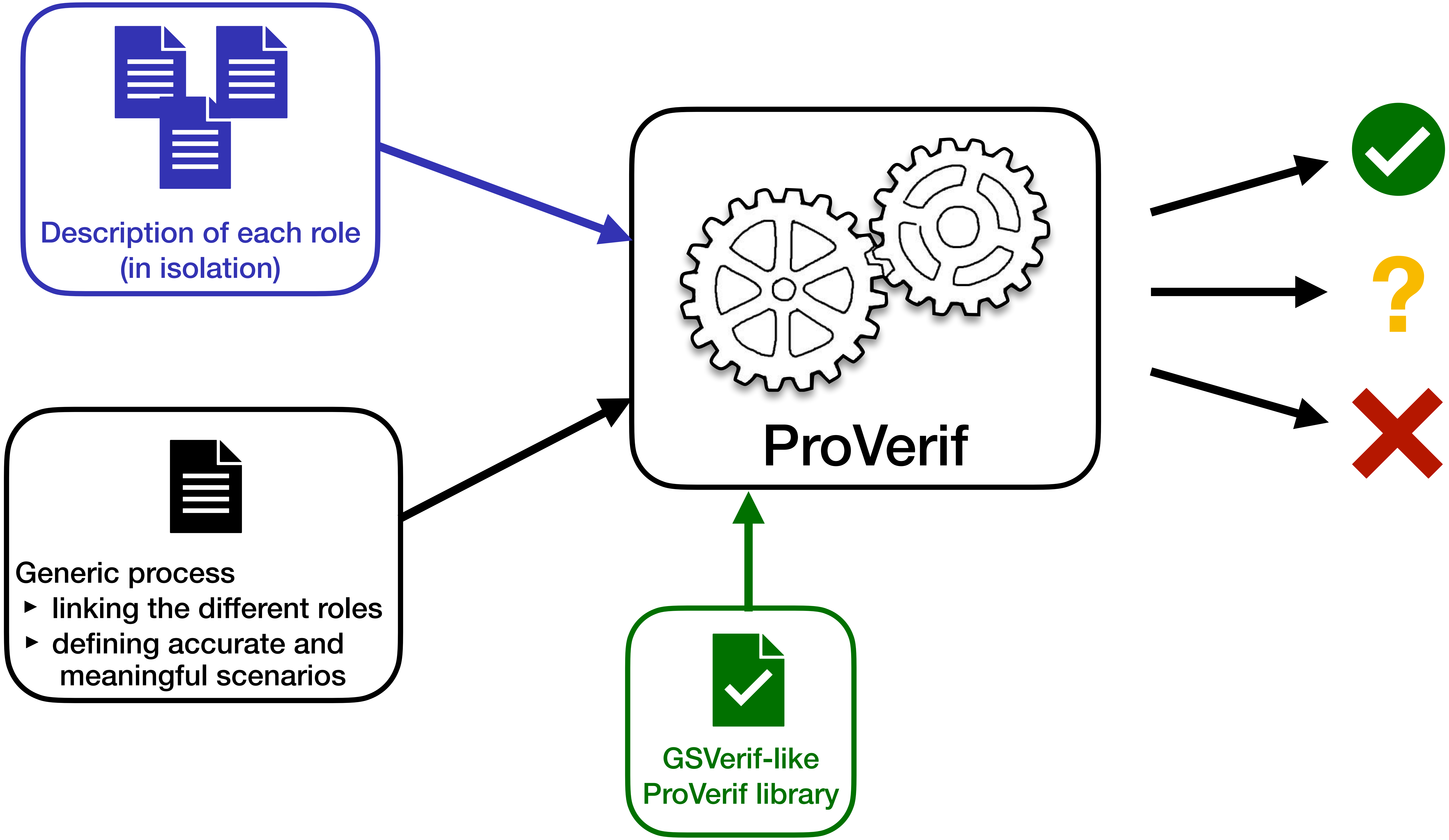
ProVerif

✓

?

✗

Our framework



Description of each role
(in isolation)

Generic process
- linking the different roles
- defining accurate and meaningful scenarios

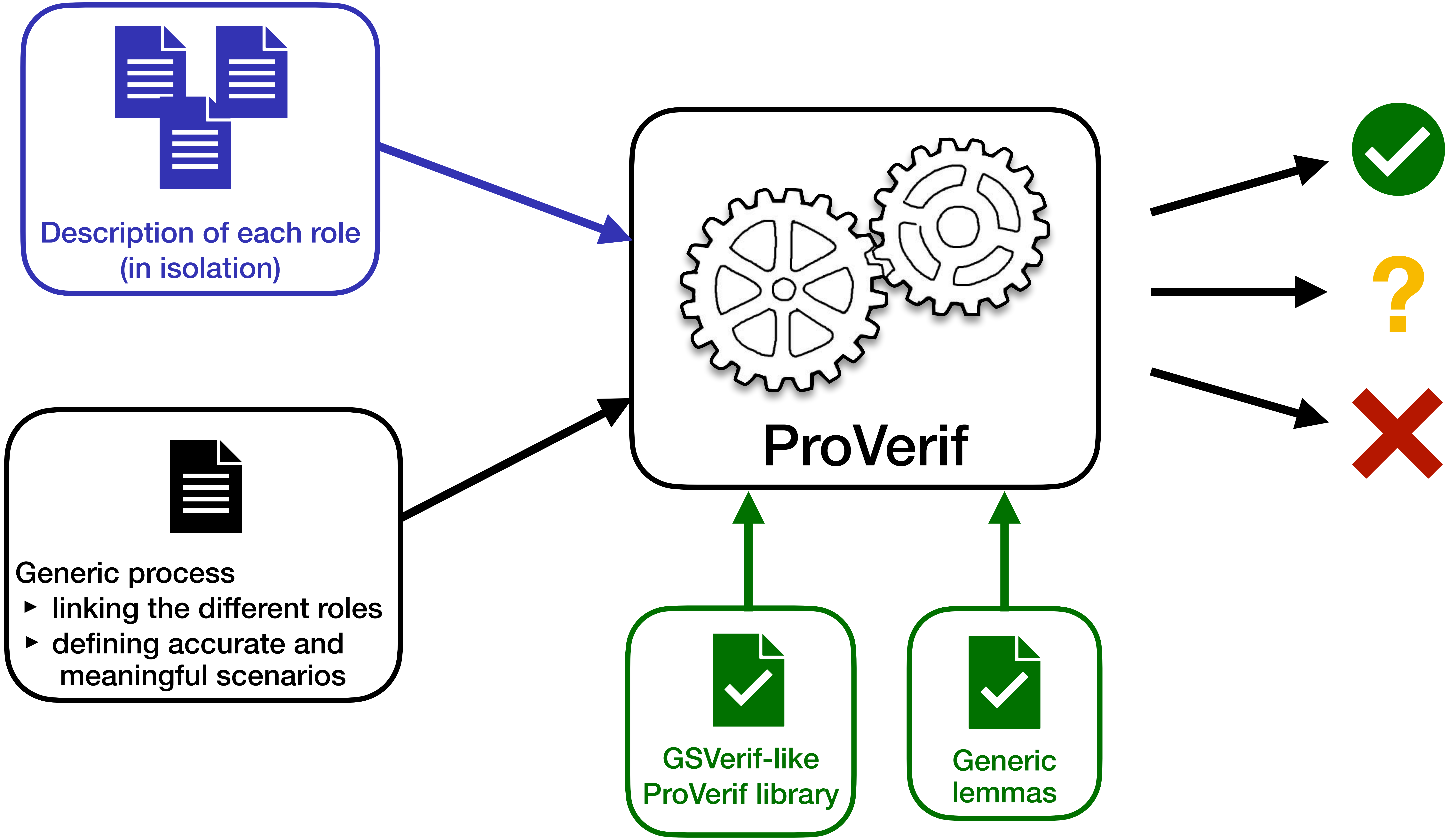
ProVerif

?

GSVerif-like
ProVerif library

[Cheval et al - CSF'18]

Our framework



Description of each role
(in isolation)

Generic process
- linking the different roles
- defining accurate and meaningful scenarios

ProVerif

✓

?

✗

GSVerif-like
ProVerif library

Generic
lemmas

[Cheval et al - CSF'18]

Details

Protocol specific processes

- **12 processes**
- **Setup phase:** 4 processes (how voting data are generated, how they are received by voters, what are their initial knowledge, what is a valid vote)
- **Voting phase:**
  - **Voter:** 2 processes (how a voter casts a vote, how they verify)
  - **Bulletin board:** 5 processes (how to update the bulletin board, what is a valid ballot, how voters are publicly identified)
- **Tally:** 1 process (how to open a ballot)

Details

Protocol specific processes

- **12 processes**
- **Setup phase:** 4 processes (how voting data are generated, how they are received by voters, what are their initial knowledge, what is a valid vote)
- **Voting phase:**
  - **Voter:** 2 processes (how a voter casts a vote, how they verify)
  - **Bulletin board:** 5 processes (how to update the bulletin board, what is a valid ballot, how voters are publicly identified)
- **Tally:** 1 process (how to open a ballot)

We do model the tally unlike previous approaches

Details

Protocol specific processes

- **12 processes**
- **Setup phase:** 4 processes (how voting data are generated, how they are received by voters, what are their initial knowledge, what is a valid vote)
- **Voting phase:**
  - **Voter:** 2 processes (how a voter casts a vote, how they verify)
  - **Bulletin board:** 5 processes (how to update the bulletin board, what is a valid ballot, how voters are publicly identified)
- **Tally:** 1 process (how to open a ballot)

We do model the tally unlike previous approaches

Generic processes and libraries

- **8 processes** (voter registration, voting process, tally, main system…)
- Unbounded number of elections and voters
- Modeler can define honesty assumptions through restrictions
- GSVerif-like axioms to manipulate cells, counters, etc
  - ➡ 2 new axioms for nested counters and emphasize term freshness
- 8 well-crafted lemmas (27 queries) to improve termination and accuracy

Applications

Protocol	Origin of the files	Voter	Registrar (setup)	Server (1 CCR/M)	E2E verifiability
Helios (toy)	(new files)	😇	—	😇	✓ 16s
Belenios (tally)	(existing personal files)	😇	😇 😈	😈 😇	✓ 24s
Belenios (last)	(existing personal files)	😇	😇	😇	✗ 5s
Belenios-counter (last)	(existing personal files)	😇	😇	😇	✗ 8s
Belenios-hash[1] (last)	(new files)	😇	😇 😈	😈 😇	✓ 62s
Swiss Post	(Swiss Post gitlab[2])	😇	😇	😇	✓ 58s
CHVote	[Bernhard et al - 2018]	😇	😇	😇	✓ 17s

[1] inspired by [Baloglu et al - EVoteID 2021]

[2] https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/Symbolic-models

Conclusion

✓ 1. **Exact characterization of E2E verifiability**

> **Theorem -** An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

✓ 2. **A ProVerif framework to analyze evoting protocols**

Applied to several protocols: Helios, Belenios, Swiss Post, CHVote

Conclusion

✓

1. Exact characterization of E2E verifiability

Theorem - An evoting protocol satisfies E2E verifiability if and only if it satisfies Query 1 and Query 2

✓

2. A ProVerif framework to analyze evoting protocols

Applied to several protocols: Helios, Belenios, Swiss Post, CHVote

Future work

- Extend the framework to analyze vote secrecy
- Extend GSVerif with the new invariants introduced in this work
- Improve the modeling of the tally:
  - consider counting functions different from the multiset of votes (e.g., Condorcet, Single Transferable Vote, d'Hondt method)
  - provide a more accurate model of the homomorphic or mixnet tally