LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Theoretical Computer Science and Theorem Proving (TCSTP)
Institute for Informatics (IFI)

Proof Composition and Refinement of Voting Systems Using Isabelle/HOL

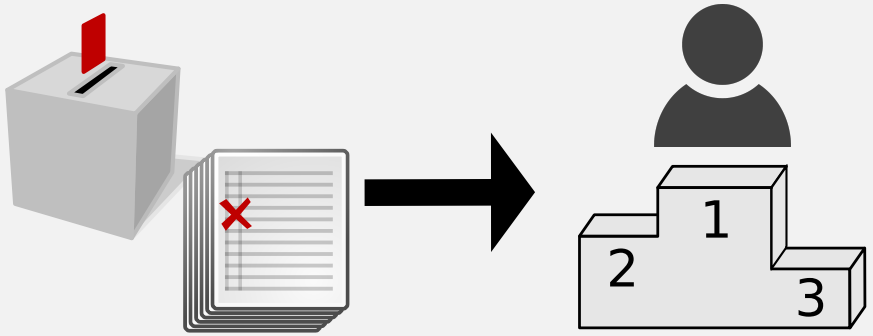
Michael Kirsten

Guest Lecture in Theorem Prover Lab at KIT, Karlsruhe

December 17, 2025

Structure of Voting Systems

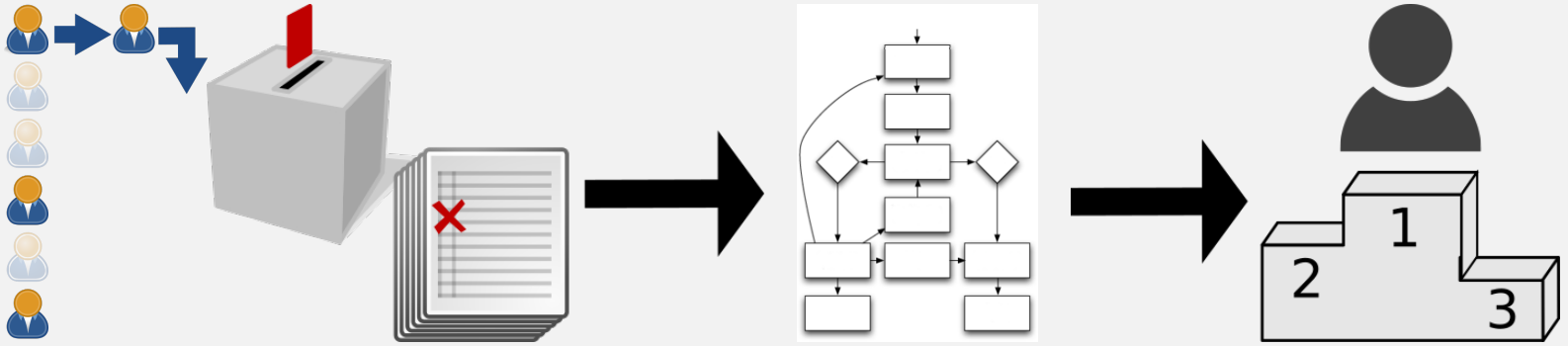
Voting System



voting process

Structure of Voting Systems

Voting System



1. management 2. casting 3. tallying

How to tally the votes?

Vote Distribution

35 %	30 %	20 %	15 %
			

Who wins the election?

How to tally the votes?

Vote Distribution			
35 %	30 %	20 %	15 %
			

Who wins the election?

- Simple plurality: 

How to tally the votes?

	Vote Distribution			
	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

Who wins the election?

- Simple plurality: 

The will of the voters might be more than a “single choice”:

How to tally the votes?

	Vote Distribution			
	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

Who wins the election?

- Simple plurality: 

The will of the voters might be more than a “single choice”:

- 65 % are dissatisfied with  .

How to tally the votes?

	Vote Distribution			
	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

Who wins the election?

- Simple plurality: 

The will of the voters might be more than a “single choice”:

- 65 % are dissatisfied with  .
- 30 % would have an advantage if they switched to  .

How to tally the votes?

Vote Distribution

	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

Who wins the election?

- Simple plurality: 

The will of the voters might be more than a “single choice”:

- 65 % are dissatisfied with  .
- 30 % would have an advantage if they switched to  .
- Then, 55 % could “cheat” with  , however.

How to tally the votes?

	Vote Distribution			
	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

Who wins the election?

- Simple plurality: 

The will of the voters might be more than a “single choice”:

- 65 % are dissatisfied with  .
- 30 % would have an advantage if they switched to  .
- Then, 55 % could “cheat” with  , however.
- Two examples of **strategic (election) manipulation**.

How to tally the votes?

Vote Distribution

	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

Free and **equal** elections should ensure that . . .

- voters express their true will without distortion, and
- every vote counts the same.

Who wins the election?

- Simple plurality: 

The will of the voters might be more than a “single choice”:

- 65 % are dissatisfied with  .
- 30 % would have an advantage if they switched to  .
- Then, 55 % could “cheat” with  , however.
- Two examples of **strategic (election) manipulation**.

Example Criterion Against Strategies: Condorcet

Definition (Condorcet Criterion)

We ought to take the Condorcet winner as sole winner if it exists.

- *a beats b* if and only if more than half the voters prefer *a* to *b*.
- *a* is a *Condorcet winner* if and only if *a* beats every other alternative.

Example Criterion Against Strategies: Condorcet

Definition (Condorcet Criterion)

We ought to take the Condorcet winner as sole winner if it exists.

- *a beats b* if and only if more than half the voters prefer *a* to *b*.
- *a* is a *Condorcet winner* if and only if *a* beats every other alternative.

	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

- Who is the Condorcet winner?

Example Criterion Against Strategies: Condorcet

Definition (Condorcet Criterion)

We ought to take the Condorcet winner as sole winner if it exists.

- *a beats b* if and only if more than half the voters prefer *a* to *b*.
- *a* is a *Condorcet winner* if and only if *a* beats every other alternative.

	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

- Who is the Condorcet winner?  .

Example Criterion Against Strategies: Condorcet

Definition (Condorcet Criterion)

We ought to take the Condorcet winner as sole winner if it exists.

- *a beats b* if and only if more than half the voters prefer *a* to *b*.
- *a* is a *Condorcet winner* if and only if *a* beats every other alternative.

	35 %	30 %	20 %	15 %
1.				
2.				
3.				
4.				
5.				

- Who is the Condorcet winner?  .

Problem

Sometimes there is no Condorcet winner.

$\Rightarrow$ *Condorcet paradox*

How can we ensure such properties?

Research Approach

- Proven construction of tallying algorithms based on *basic* modules
- Algorithm becomes comprehensible by such a construction
- Proof by application of proven *construction rules* $\Rightarrow$ **Composition of rules is explanation**
- Construction of tallying algorithms with desired properties (and explanation) can be automated

How can we ensure such properties?

Research Approach

- Proven construction of tallying algorithms based on *basic* modules
- Algorithm becomes comprehensible by such a construction
- Proof by application of proven *construction rules* $\Rightarrow$ **Composition of rules is explanation**
- Construction of tallying algorithms with desired properties (and explanation) can be automated

Example for Condorcet Criterion

Base function s: For any alternative, compute sum of all less-preferred alternatives for each vote

How can we ensure such properties?

Research Approach

- Proven construction of tallying algorithms based on *basic* modules
- Algorithm becomes comprehensible by such a construction
- Proof by application of proven *construction rules* $\Rightarrow$ **Composition of rules is explanation**
- Construction of tallying algorithms with desired properties (and explanation) can be automated

Example for Condorcet Criterion

Base function s: For any alternative, compute sum of all less-preferred alternatives for each vote

Example Construction: $(m_s \circlearrowleft_t)$

- m_s eliminates alternative with lowest value for s

How can we ensure such properties?

Research Approach

- Proven construction of tallying algorithms based on *basic* modules
- Algorithm becomes comprehensible by such a construction
- Proof by application of proven *construction rules* $\Rightarrow$ **Composition of rules is explanation**
- Construction of tallying algorithms with desired properties (and explanation) can be automated

Example for Condorcet Criterion

Base function s: For any alternative, compute sum of all less-preferred alternatives for each vote

Example Construction: $(m_s \circlearrowleft_t)$

- m_s eliminates alternative with lowest value for s $\Rightarrow$ **Never eliminates Condorcet winner**

How can we ensure such properties?

Research Approach

- Proven construction of tallying algorithms based on *basic* modules
- Algorithm becomes comprehensible by such a construction
- Proof by application of proven *construction rules* $\Rightarrow$ **Composition of rules is explanation**
- Construction of tallying algorithms with desired properties (and explanation) can be automated

Example for Condorcet Criterion

Base function s: For any alternative, compute sum of all less-preferred alternatives for each vote

Example Construction: $(m_s \circlearrowleft_t)$

- m_s eliminates alternative with lowest value for s $\Rightarrow$ **Never eliminates Condorcet winner**
- $\circlearrowleft_t$ repeats module until only one alternative remains

How can we ensure such properties?

Research Approach

- Proven construction of tallying algorithms based on *basic* modules
- Algorithm becomes comprehensible by such a construction
- Proof by application of proven *construction rules* $\Rightarrow$ **Composition of rules is explanation**
- Construction of tallying algorithms with desired properties (and explanation) can be automated

Example for Condorcet Criterion

Base function s: For any alternative, compute sum of all less-preferred alternatives for each vote

Example Construction: $(m_s \circlearrowleft_t)$

- m_s eliminates alternative with lowest value for s $\Rightarrow$ **Never eliminates Condorcet winner**
- $\circlearrowleft_t$ repeats module until only one alternative remains $\Rightarrow$ **Condorcet winner wins if it exists**

Formal Robust-by-Construction Framework

Formal Methods

- Theorem prover for human-readable and machine-checked proofs
- Translation of mathematical functions into verified Scala/Haskell programs

Formal Robust-by-Construction Framework

Formal Methods

- Theorem prover for human-readable and machine-checked proofs
- Translation of mathematical functions into verified Scala/Haskell programs
- Simple logic programming based on linear resolution
- Automatic search on facts for "closed world assumption"

Formal Robust-by-Construction Framework

Formal Methods

- Theorem prover for human-readable and machine-checked proofs
- Translation of mathematical functions into verified Scala/Haskell programs
- Simple logic programming based on linear resolution
- Automatic search on facts for "closed world assumption"

- Formalization and proofs of modules and robustness (composition) rules
- Library of components with verified robustness properties
- Automatic (source-)code generation of composed procedure
- Translation of proven rules to Prolog for automatic search to construct proofs:

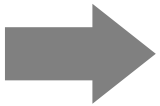
Formal Robust-by-Construction Framework

Formal Methods

- Theorem prover for human-readable and machine-checked proofs
- Translation of mathematical functions into verified Scala/Haskell programs
- Simple logic programming based on linear resolution
- Automatic search on facts for "closed world assumption"

- Formalization and proofs of modules and robustness (composition) rules
- Library of components with verified robustness properties
- Automatic (source-)code generation of composed procedure
- Translation of proven rules to Prolog for automatic search to construct proofs:

```
theorem condorcet_consistent_seq1:
  assumes condorcet_m: "condorcet_consistent m" and
          module_n:    "electoral_module n"
  shows "condorcet_consistent (m ▷ n)"
```



```
condorcetconsistent(seqcomp(X,Y)):-
    condorcetconsistent(X), module(Y).
```

Formal Robust-by-Construction Framework

Formal Methods

- Theorem prover for human-readable and machine-checked proofs
- Translation of mathematical functions into verified Scala/Haskell programs
- Simple logic programming based on linear resolution
- Automatic search on facts for "closed world assumption"

- Formalization and proofs of modules and robustness (composition) rules
- Library of components with verified robustness properties
- Automatic (source-)code generation of composed procedure
- Translation of proven rules to Prolog for automatic search to construct proofs:

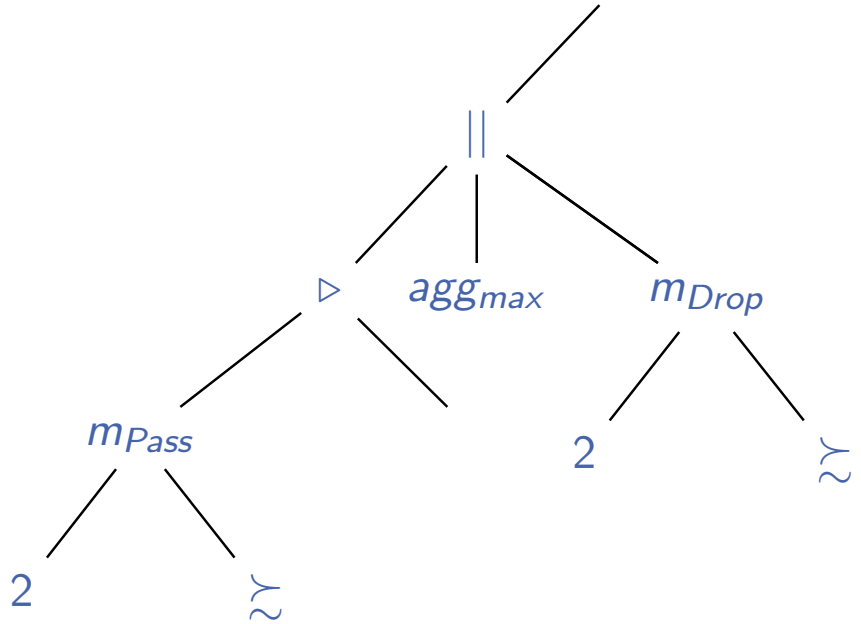
```
?- condorcetconsistent(X), electing(X), monotone(X).
  X = seqcomp(m_cond, m_borda) ;
  X = seqcomp(m_cond, m_plurality) ;
  X = seqcomp(m_cond, seqcomp(m_cond, m_borda)) .
```

Example: Sequential Majority Comparison

Example: Sequential Majority Comparison

$((m_{Pass}(2) \triangleright \quad) \,||_{agg_{max}}\, m_{Drop}(2))$

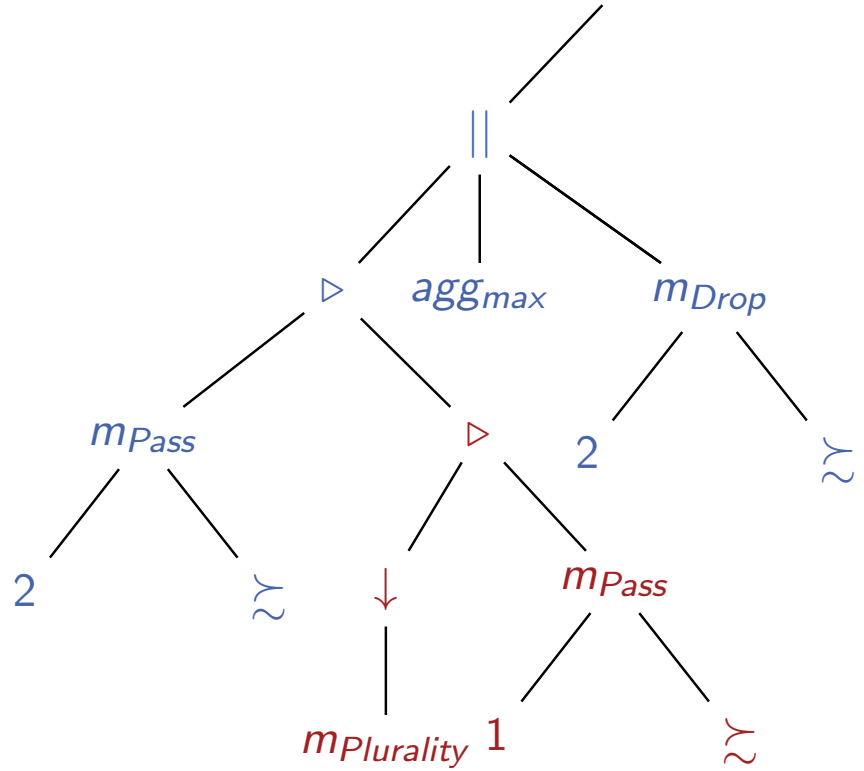
- Pick **two** candidates.



Example: Sequential Majority Comparison

$$((m_{Pass}(2) \triangleright m_{Plurality}{\downarrow} \triangleright m_{Pass}(1)) \,||_{agg_{max}}\, m_{Drop}(2))$$

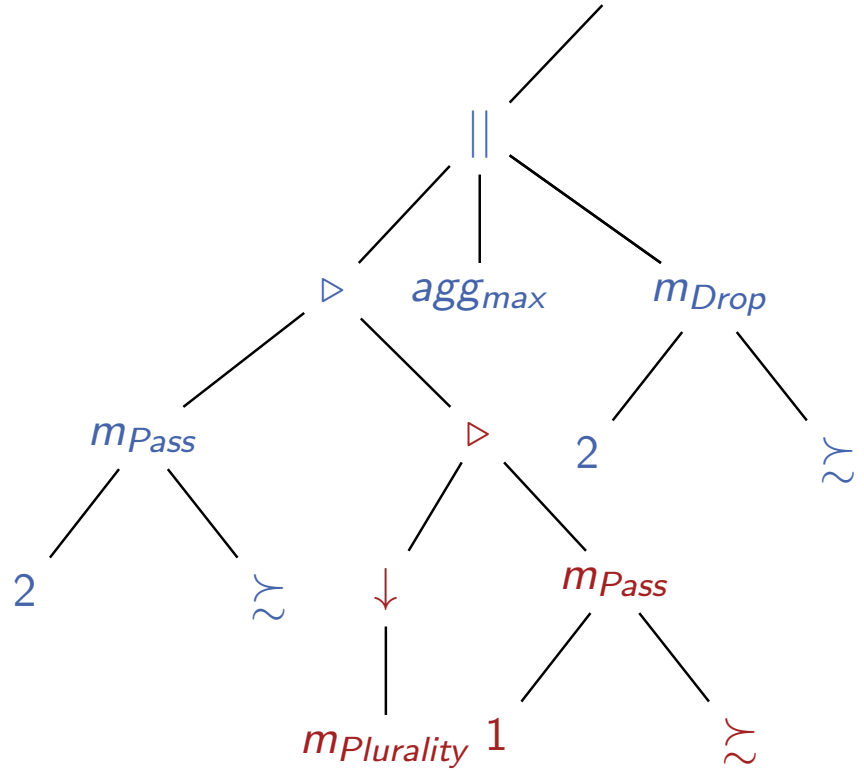
- Pick **two** candidates.
  - Choose **winner**.



Example: Sequential Majority Comparison

$((m_{Pass}(2) \triangleright m_{Plurality}\downarrow \triangleright m_{Pass}(1)) \,||_{agg_{max}}\, m_{Drop}(2))$

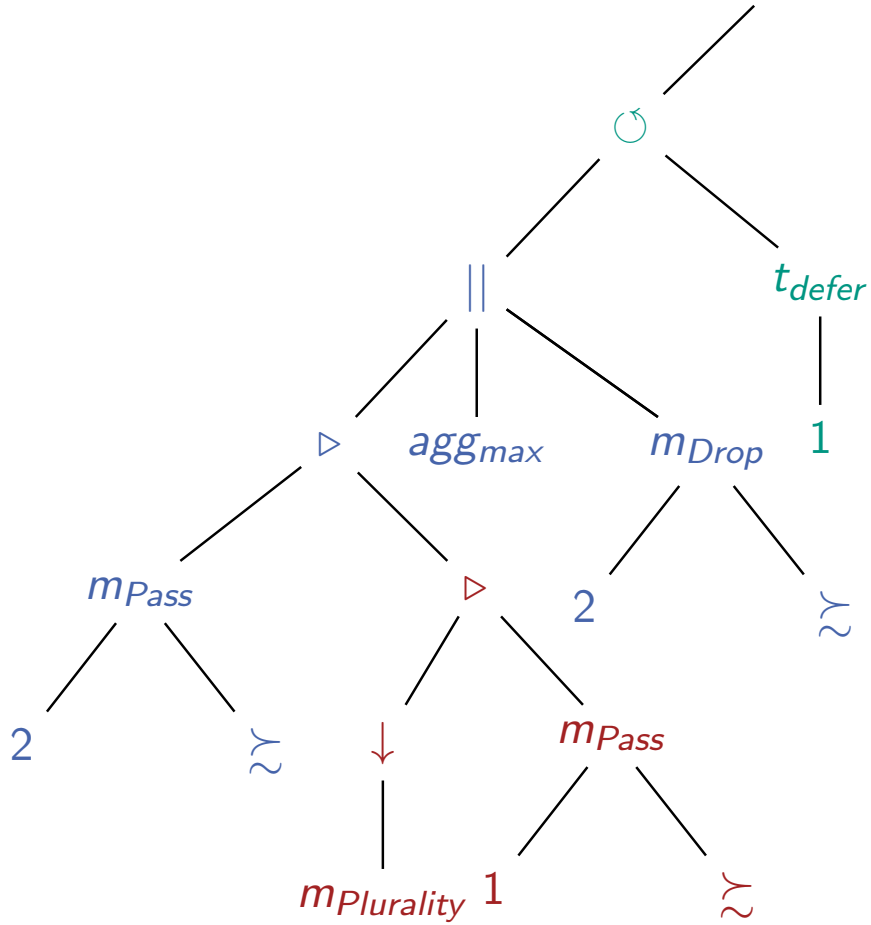
- Pick **two** candidates.
  - Choose **winner**.
  - Find new opponent.



Example: Sequential Majority Comparison

$$((m_{Pass}(2) \triangleright m_{Plurality}{\downarrow} \triangleright m_{Pass}(1)) \,||_{agg_{max}}\, m_{Drop}(2))\,\circlearrowleft_{|d|=1}$$

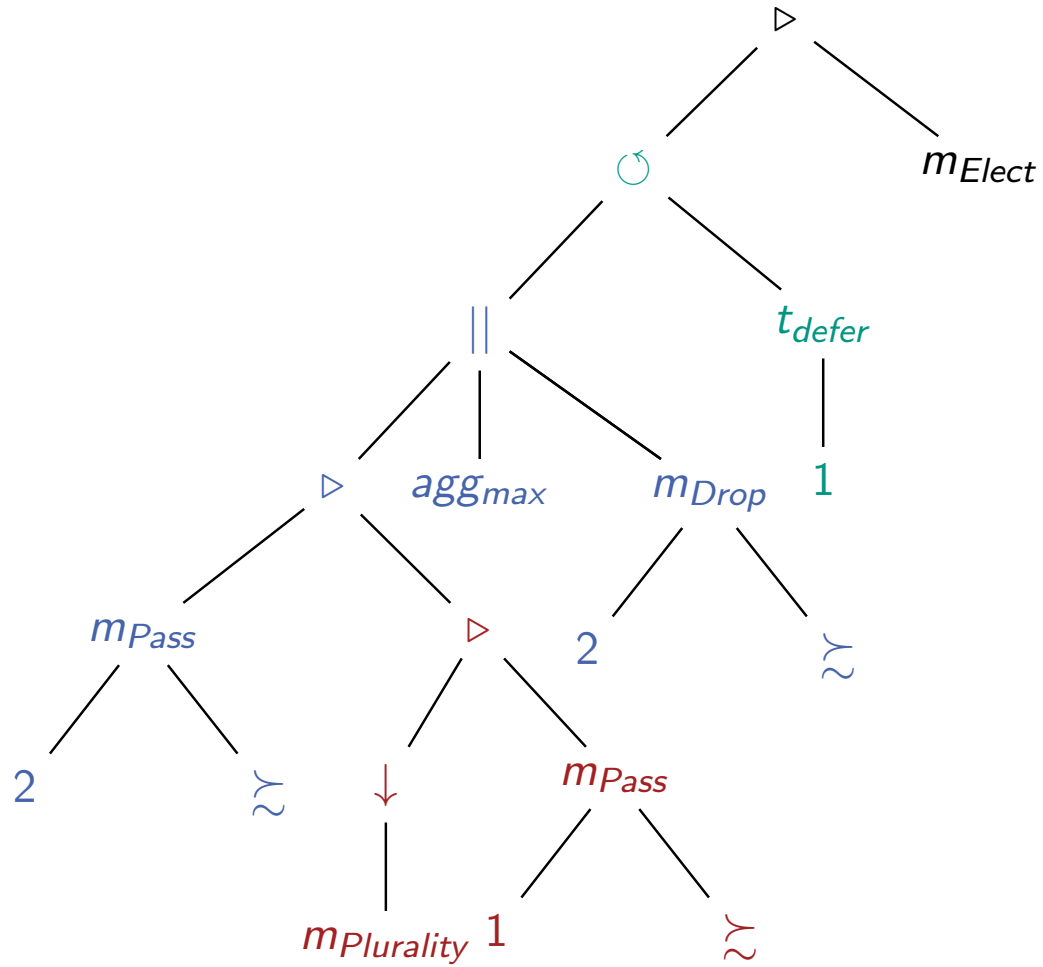
- Pick **two** candidates.
  - Choose **winner**.
  - Find new opponent.
  - **Repeat**.



Example: Sequential Majority Comparison

$$((m_{Pass}(2) \triangleright m_{Plurality}{\downarrow} \triangleright m_{Pass}(1)) \,||_{agg_{max}}\, m_{Drop}(2))\,\circlearrowleft_{|d|=1} \triangleright m_{Elect}$$

- Pick **two** candidates.
  - Choose **winner**.
  - Find new opponent.
  - **Repeat**.



Proof Composition for Reliable Voting Rules

Proof of Monotonicity for Sequential Majority Comparison

$m_{elim} \circlearrowleft_{|d|=1} \triangleright m_{elect}$ is monotone

Proof Composition for Reliable Voting Rules

Proof of Monotonicity for Sequential Majority Comparison

$$\frac{m_{elect} \text{ is electing} \quad m_{elim}\circlearrowleft_{|d|=1} \text{ is non-electing} \quad m_{elim}\circlearrowleft_{|d|=1} \text{ defers 1 alternative} \quad m_{elim}\circlearrowleft_{|d|=1} \text{ is defer-lift-invariant}}{m_{elim}\circlearrowleft_{|d|=1} \triangleright m_{elect} \text{ is monotone}}$$

Proof Composition for Reliable Voting Rules

Proof of Monotonicity for Sequential Majority Comparison

$$\dfrac{\dfrac{m_{elim} \text{ is non-electing} \quad m_{elim} \text{ eliminates 1 alternative}}{m_{elim}\circlearrowleft_{|d|=1} \text{ defers 1 alternative}} \quad \dfrac{m_{elect} \text{ is electing} \quad m_{elim}\circlearrowleft_{|d|=1} \text{ is non-electing} \quad m_{elim}\circlearrowleft_{|d|=1} \text{ is defer-lift-invariant}}{}}{m_{elim}\circlearrowleft_{|d|=1} \rhd m_{elect} \text{ is monotone}}$$

Proof Composition for Reliable Voting Rules

Proof of Monotonicity for Sequential Majority Comparison

$$\dfrac{\dfrac{\dfrac{\vdots}{m_{elim} \text{ is non-electing}}}{m_{elim} \text{ eliminates 1 alternative}} \qquad \dfrac{\dfrac{\vdots}{m_{elect} \text{ is electing}}}{m_{elim}\circlearrowleft_{|d|=1} \text{ is non-electing}}}{\dfrac{m_{elim}\circlearrowleft_{|d|=1} \text{ defers 1 alternative} \qquad m_{elim}\circlearrowleft_{|d|=1} \text{ is defer-lift-invariant}}{m_{elim}\circlearrowleft_{|d|=1} \triangleright m_{elect} \text{ is monotone}}}$$

Proof Composition for Reliable Voting Rules

Proof of Monotonicity for Sequential Majority Comparison

$$\dfrac{\dfrac{\dfrac{\vdots}{m_{elim} \text{ is non-electing}}}{m_{elim} \text{ eliminates 1 alternative}} \quad \dfrac{\dfrac{\vdots}{m_{elect} \text{ is electing}}}{m_{elim}\circlearrowleft_{|d|=1} \text{ is non-electing}}}{\dfrac{m_{elim}\circlearrowleft_{|d|=1} \text{ defers 1 alternative} \quad m_{elim}\circlearrowleft_{|d|=1} \text{ is defer-lift-invariant}}{m_{elim}\circlearrowleft_{|d|=1} \rhd m_{elect} \text{ is monotone}}}$$

All proven within Isabelle/HOL:

- 4 basic electoral modules
- 4 compositional structures
- 19 composition rules (10 reusable lemmas)
- 13 module properties

Proof Composition for Reliable Voting Rules

Proof of Monotonicity for Sequential Majority Comparison

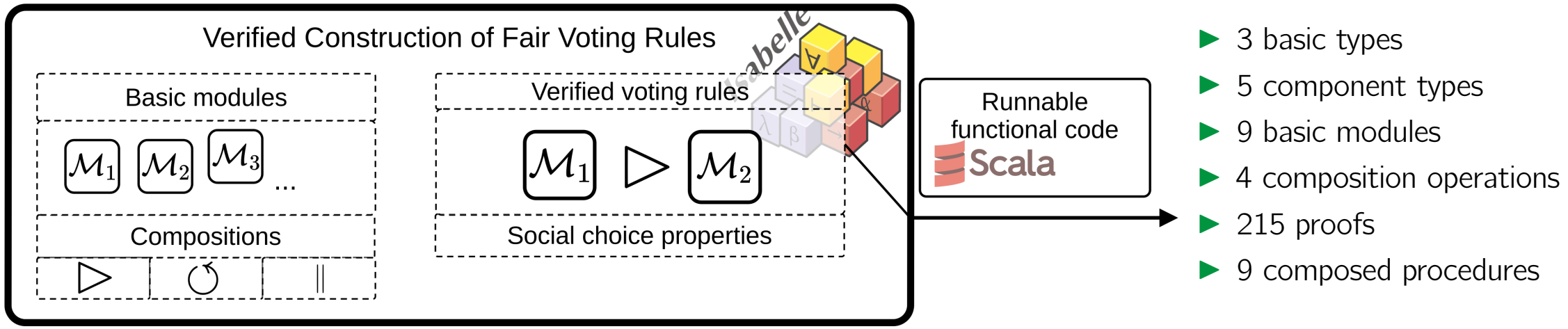
$$\dfrac{\dfrac{\vdots}{\dfrac{m_{elim}\text{ is non-electing}}{m_{elim}\text{ eliminates 1 alternative}}}}{m_{elim}\circlearrowleft_{|d|=1}\text{ defers 1 alternative}} \qquad \dfrac{\vdots}{\dfrac{m_{elect}\text{ is electing}}{m_{elim}\circlearrowleft_{|d|=1}\text{ is non-electing}}}}{m_{elim}\circlearrowleft_{|d|=1}\text{ is defer-lift-invariant}}$$

$$\dfrac{\text{defers 1 alternative} \qquad \text{is defer-lift-invariant}}{m_{elim}\circlearrowleft_{|d|=1} \triangleright m_{elect}\text{ is monotone}}$$

All proven within Isabelle/HOL:

- 4 basic electoral modules
- 4 compositional structures
- 19 composition rules (10 reusable lemmas)
- 13 module properties

Framework for Correctness-by-Construction of Voting Rules



- 3 basic types
- 5 component types
- 9 basic modules
- 4 composition operations
- 215 proofs
- 9 composed procedures

Now Some Abstraction: The Elimination Set

Evaluation Function

- Idea: Assign an alternative's value by setting them into relation to the others
- Example: Minimax Score $m(a, A, p)$, the worst that alternative a does against any other alternative

Now Some Abstraction: The Elimination Set

Evaluation Function

- Idea: Assign an alternative's value by setting them into relation to the others
- Example: Minimax Score $m(a, A, p)$, the worst that alternative a does against any other alternative

Elimination Set

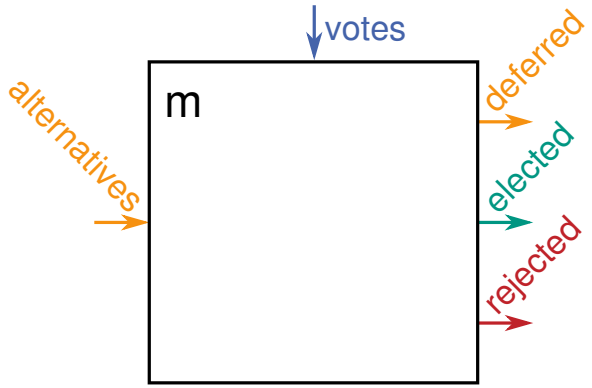
Eliminate each alternative whose value is in relation r with threshold value t.

$$E = \{a \in A \ : \ (e(a, A, p), t) \in r\} \text{ with}$$

- Eligible alternatives A, profile p
- Evaluation function e
- Threshold value $t \in \mathbb{N}$
- Binary relation r on $\mathbb{N}$

Making it a Module: The Elimination Module

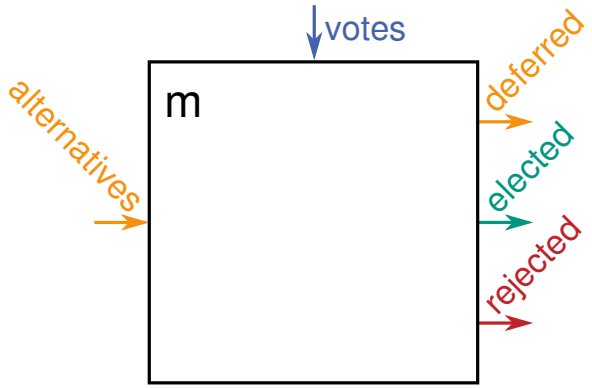
Basic Module Type Inspired by **Grilli di Cortona et al. (1999)**



- Segmentation of outcome into set triple for further processing
- Common module type for general composition operations

Making it a Module: The Elimination Module

Basic Module Type Inspired by Grilli di Cortona et al. (1999)



- Segmentation of outcome into set triple for further processing
- Common module type for general composition operations

Elimination Module

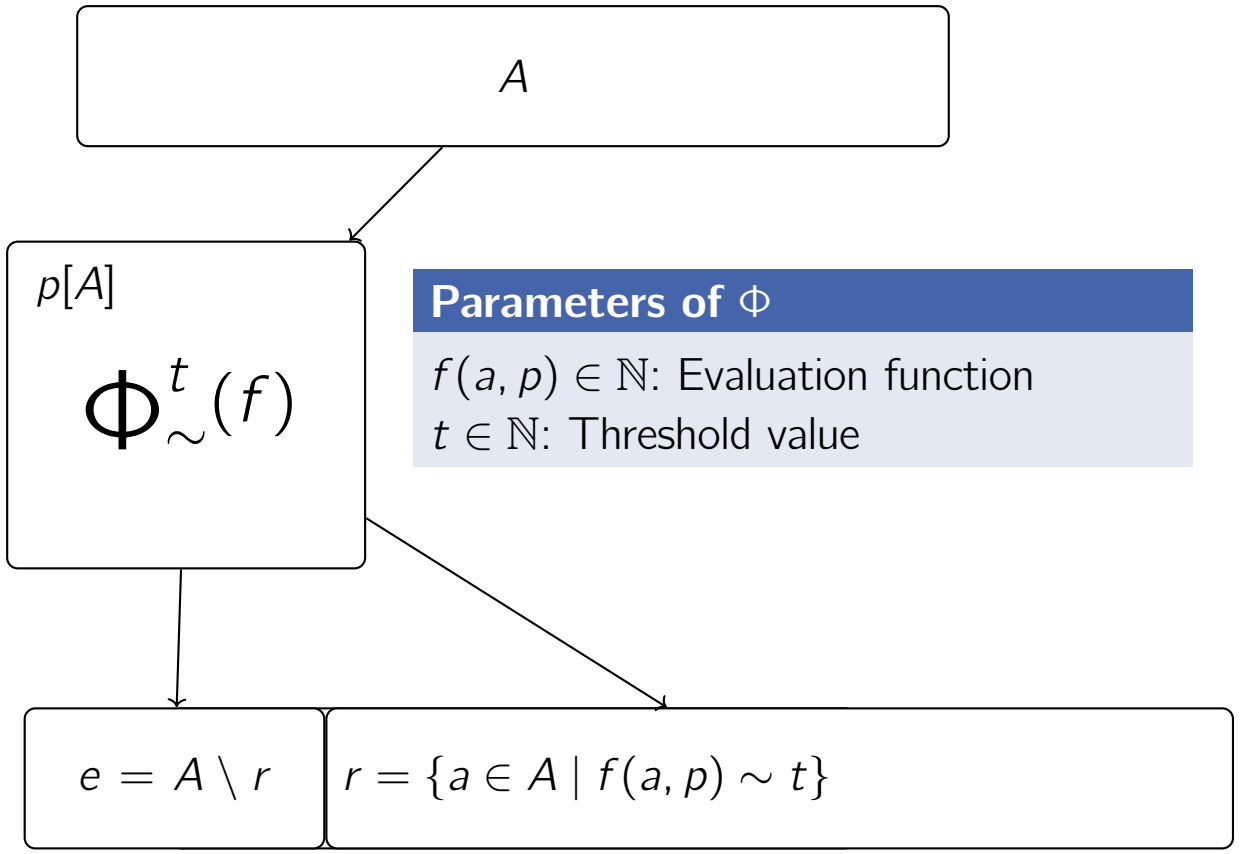
Reject elimination set if and only if we would not reject all alternatives.

$$m(A, p) := \begin{cases} (\varnothing, E, A \setminus E) & \text{if } E \subsetneq A \\ (\varnothing, \varnothing, A) & \text{else} \end{cases}$$

Special Cases

- LESS Eliminator: $a \in E \iff a$'s value $<$ threshold value
- MAX Eliminator: $a \in E \iff a$'s value $<$ maximum value of all alternatives

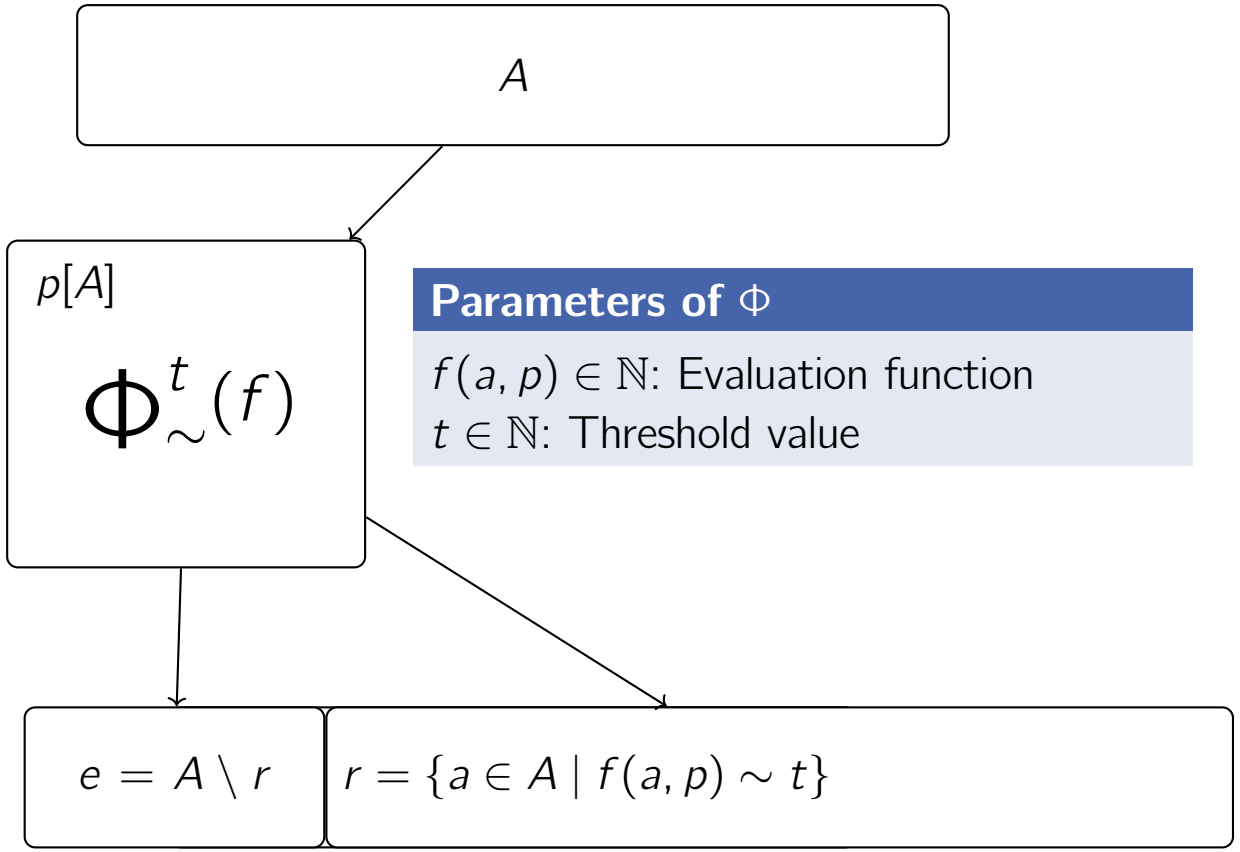
Turning the Elimination Module Into a Program



Parameters of Φ

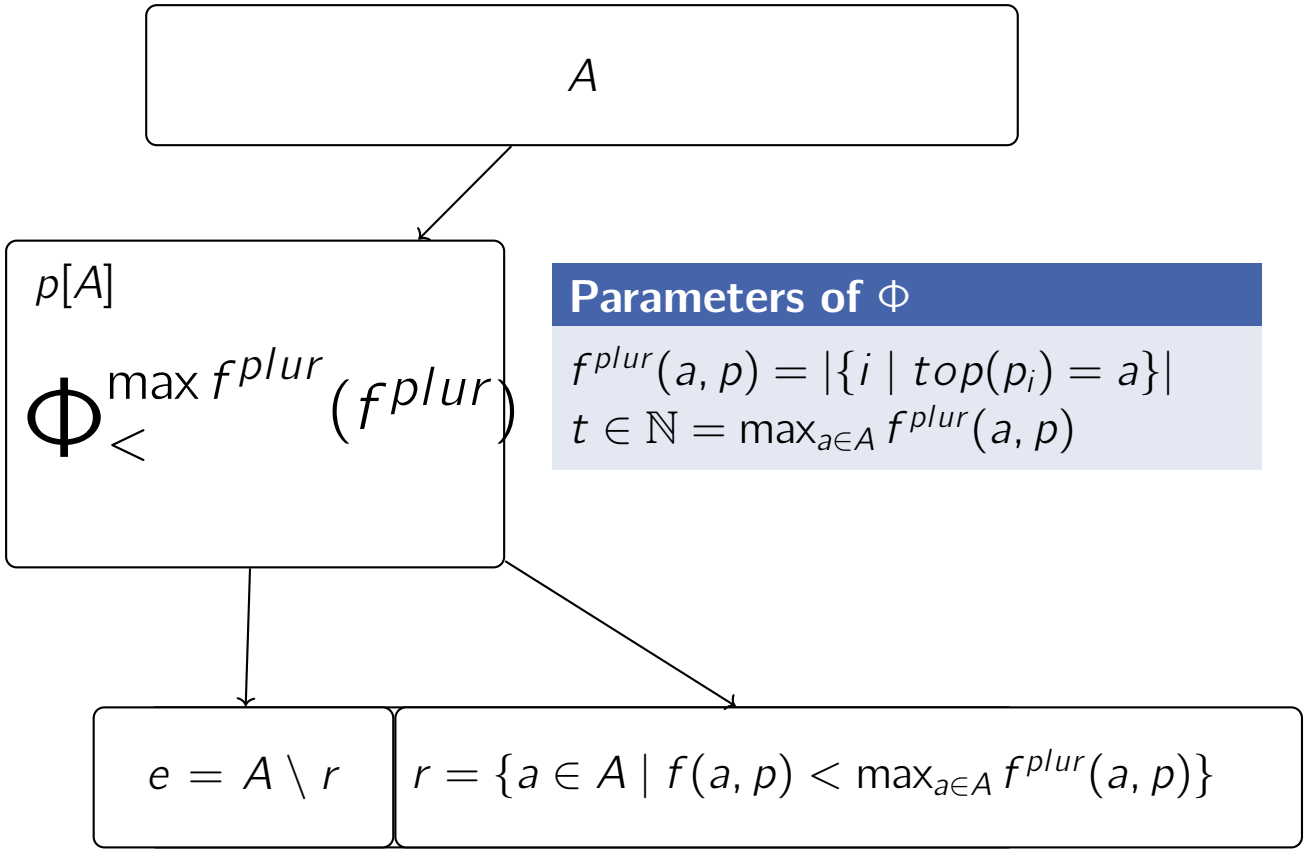
$f(a,p) \in \mathbb{N}$: Evaluation function
$t \in \mathbb{N}$: Threshold value

Turning the Elimination Module Into a Program



foreach $a \in A$ **do**
  if $f(a,p) \sim t$ **then** $r \leftarrow r \cup \{a\}$;
  else $e \leftarrow e \cup \{a\}$;
return (e, r);

Refined Elimination Module for the Plurality Rule



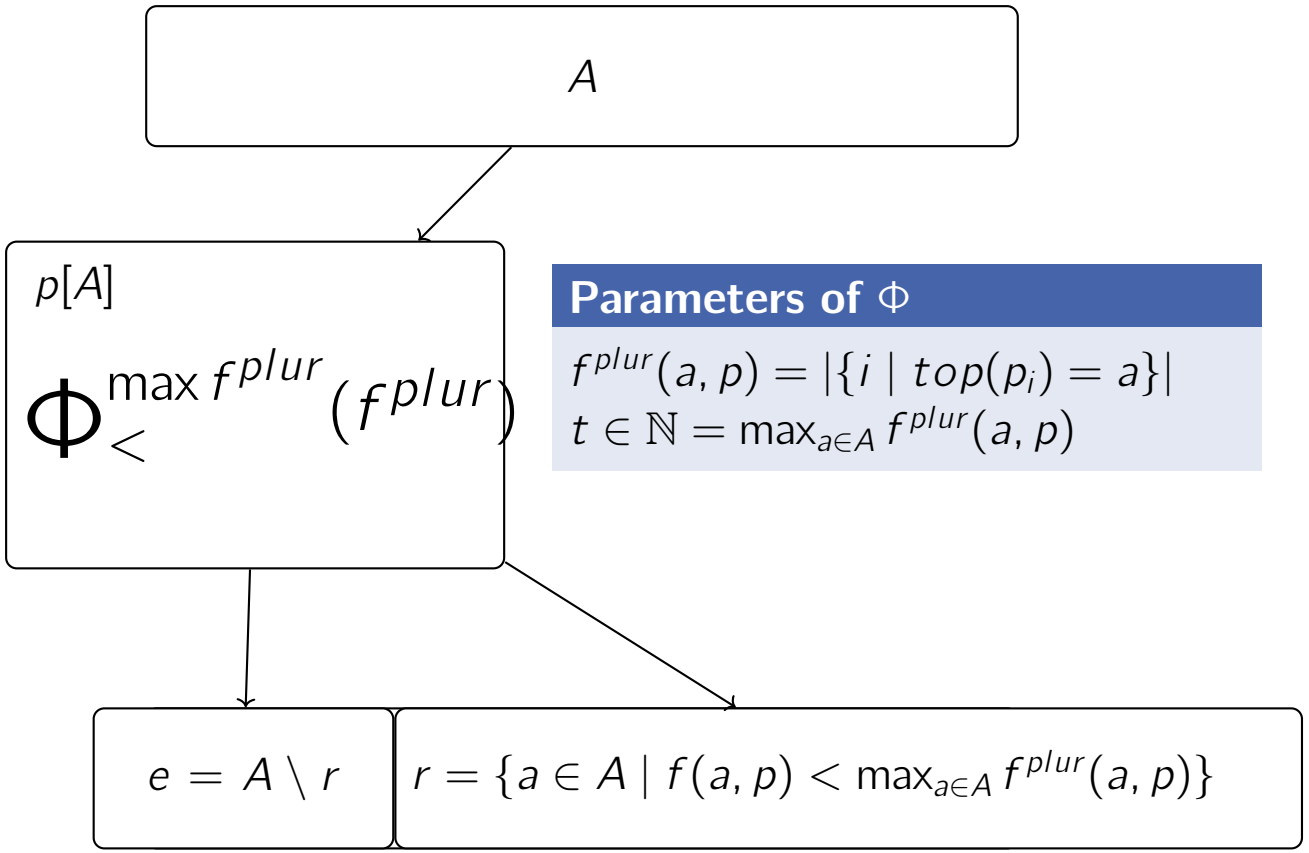
Parameters of Φ

$f^{plur}(a,p) = |\{i \mid top(p_i) = a\}|$

$t \in \mathbb{N} = \max_{a \in A} f^{plur}(a,p)$

$t \leftarrow \max_{a \in A} f^{plur}(a,p)$;
foreach $a \in A$ **do**
  if $f^{plur}(a,p) < t$ **then** $r \leftarrow r \cup \{a\}$;
  else $e \leftarrow e \cup \{a\}$;
return (e,r) ;

Refined Elimination Module for the Plurality Rule



Parameters of Φ

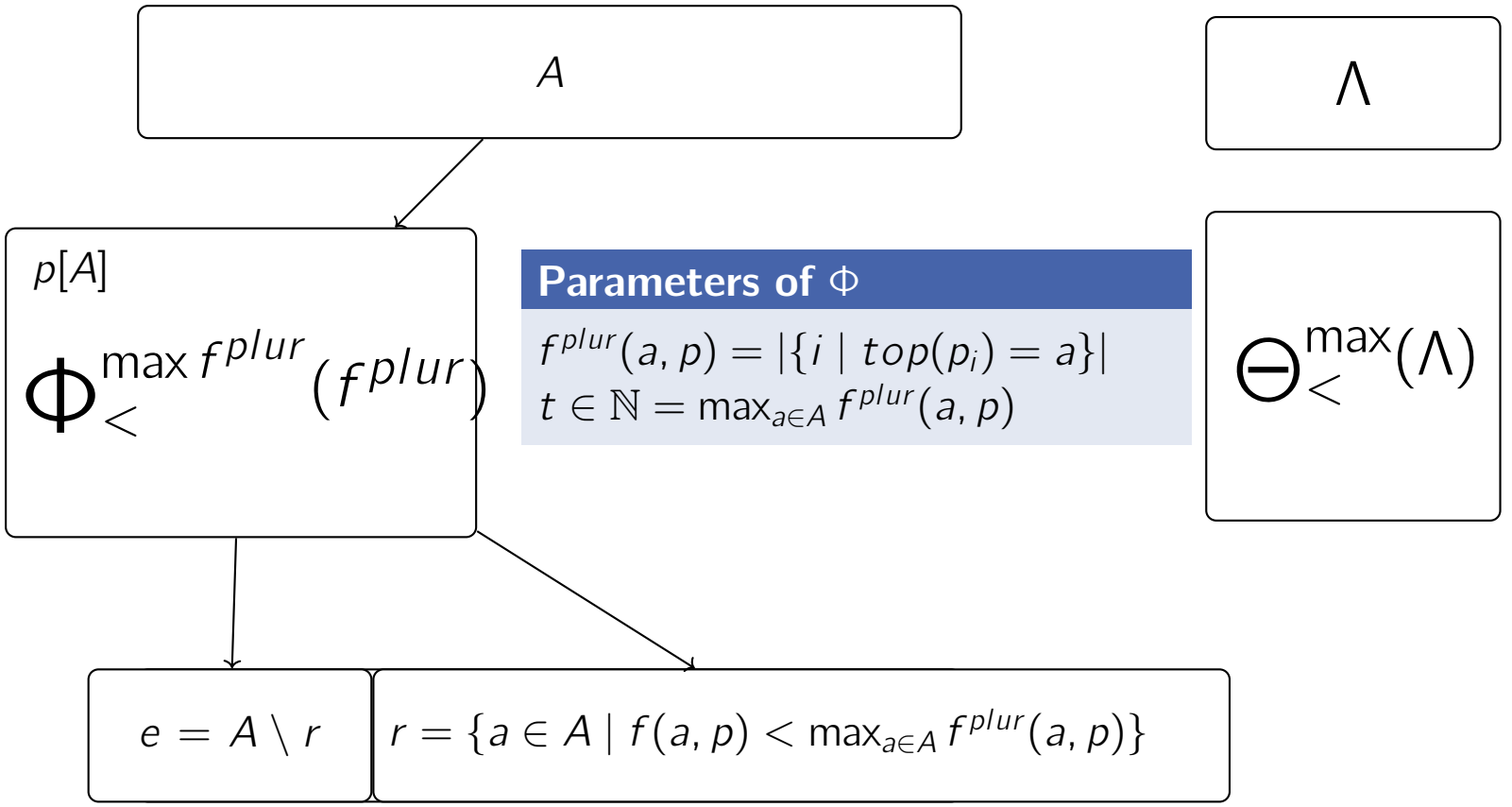
$f^{plur}(a,p) = |\{i \mid top(p_i) = a\}|$

$t \in \mathbb{N} = \max_{a \in A} f^{plur}(a,p)$

$map : \tau_{alt} \to \mathbb{N}$;
for $a \in A$ **do** $map[a] \leftarrow f^{plur}(a,p)$;

assert $\forall a \in A\ map[a] = f^{plur}(a,p)$
$t \leftarrow \texttt{max}(map[:])$;
foreach $a \in A$ **do**
  if $map[a] < t$ **then** $r \leftarrow r \cup \{a\}$;
  else $e \leftarrow e \cup \{a\}$;
return (e, r) ;

Refined Elimination Module for the Plurality Rule



Parameters of Φ

$f^{plur}(a,p) = |\{i \mid top(p_i) = a\}|$

$t \in \mathbb{N} = \max_{a \in A} f^{plur}(a,p)$

Λ

$\ominus^{\max}_{<}(\Lambda)$

$map : \tau_{alt} \to \mathbb{N}$;
for $a \in A$ **do** $map[a] \leftarrow f^{plur}(a,p)$;

assert $\forall a \in A\ map[a] = f^{plur}(a,p)$
$t \leftarrow \texttt{max}(map[:])$;
foreach $a \in A$ **do**
  if $map[a] < t$ **then** $r \leftarrow r \cup \{a\}$;
  else $e \leftarrow e \cup \{a\}$;
return (e, r) ;

Efficient Stepwise-Refined Programs Using Isabelle Refinement Framework

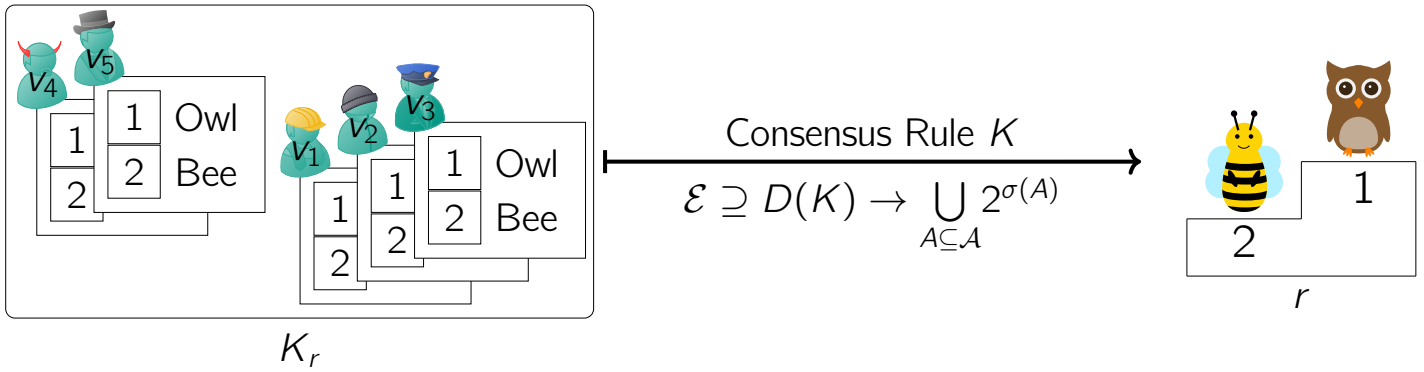
Voting Rule	#Alternatives	#Voters	Original	Refined
Plurality	9	50 000	32.0 s	0.22 s
	20	50 000	396.0 s	0.22 s
	50	50 000	✗	0.20 s
Borda	20	50 000	395.0 s	1.30 s
	20	500	4.5 s	0.12 s
Pairwise Majority	20	500	145.0 s	0.25 s
	50	500	87.0 s	5.70 s

A Construction Pattern: Distance Rationalization (DR)

- Construction of voting rules out of consensus K and election distance d (Lerer and Nitzan, 1985)
- DR rule $\mathcal{R}(K,d)(E) := \underset{r \in \sigma(E)}{\arg\min}\ d(E, K_r) = \underset{r \in \sigma(E)}{\arg\min}\ \left(\inf_{E' \in K_r} d(E, E')\right)$

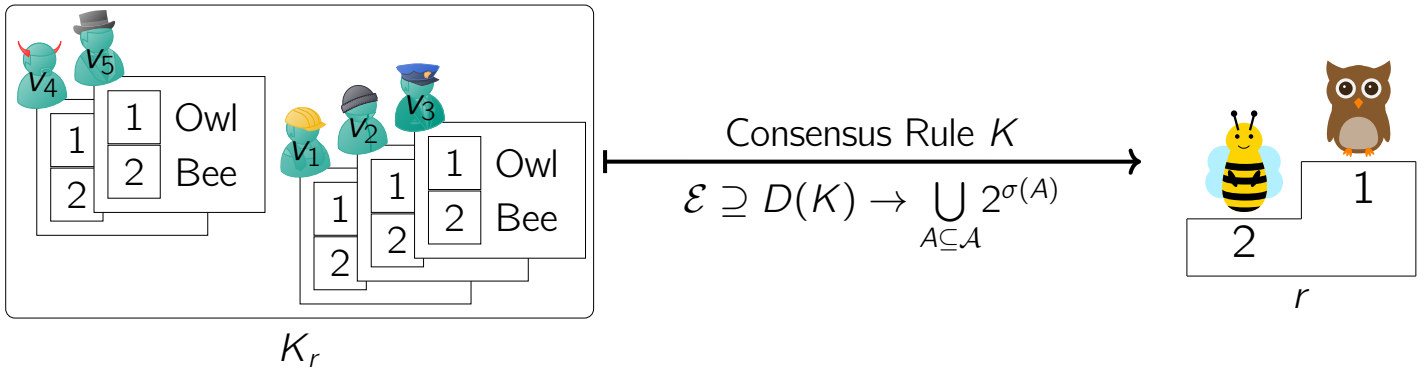
A Construction Pattern: Distance Rationalization (DR)

- Construction of voting rules out of consensus K and election distance d (Lerer and Nitzan, 1985)
- DR rule $\mathcal{R}(K,d)(E) := \underset{r \in \sigma(E)}{\arg\min}\ d(E, K_r) = \underset{r \in \sigma(E)}{\arg\min}\ \left(\inf_{E' \in K_r} d(E, E')\right)$



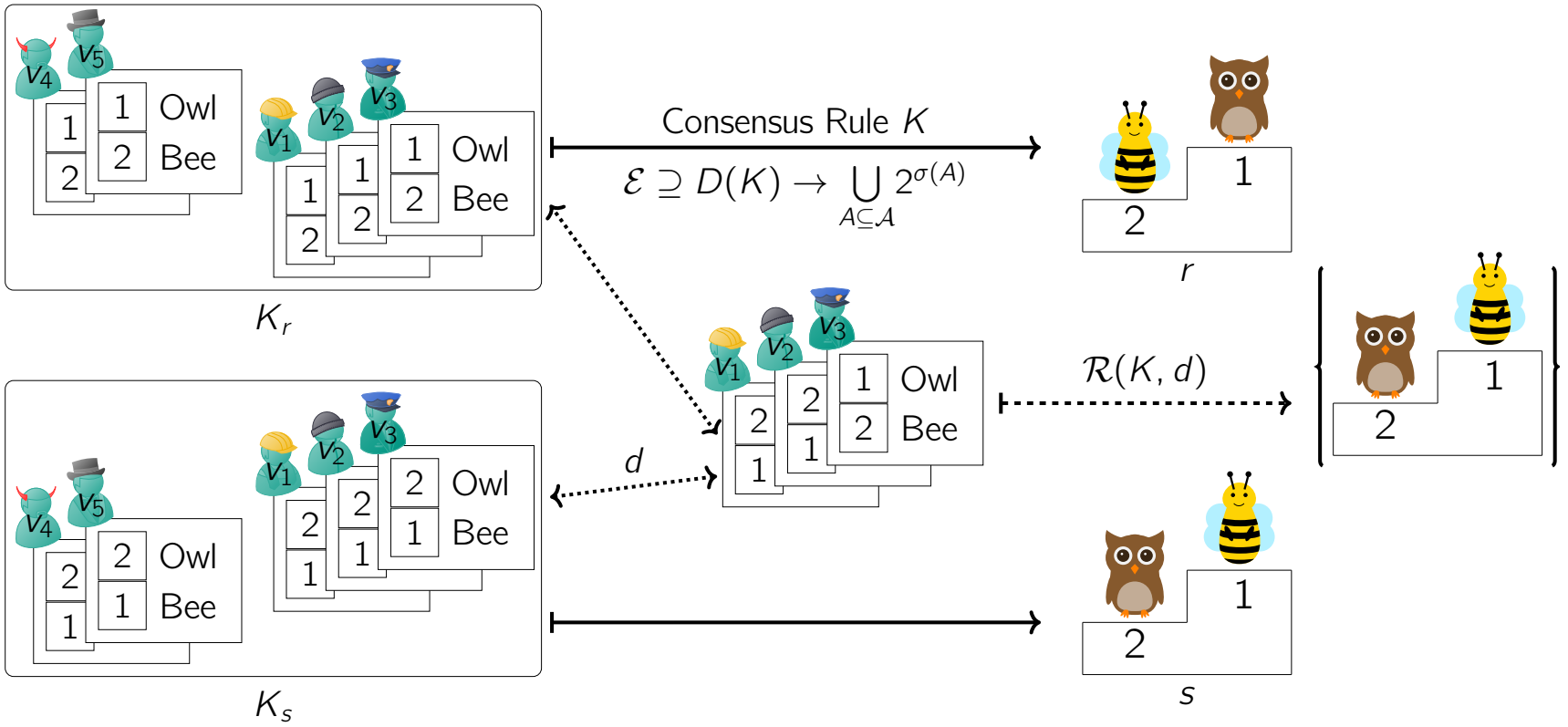
A Construction Pattern: Distance Rationalization (DR)

- Construction of voting rules out of consensus K and election distance d (Lerer and Nitzan, 1985)
- DR rule $\mathcal{R}(K,d)(E) := \underset{r \in \sigma(E)}{\arg\min}\ d(E, K_r) = \underset{r \in \sigma(E)}{\arg\min}\ \left(\inf_{E' \in K_r} d(E, E') \right)$



A Construction Pattern: Distance Rationalization (DR)

- Construction of voting rules out of consensus K and election distance d (Lerer and Nitzan, 1985)
- DR rule $\mathcal{R}(K,d)(E) := \underset{r \in \sigma(E)}{\arg\min}\ d(E, K_r) = \underset{r \in \sigma(E)}{\arg\min}\ \left(\inf_{E' \in K_r} d(E, E') \right)$



General DR-Rules for Symmetry Properties

Definition: Invariance and Equivariance

Let $f : X \to Y$, $(G, *)$ a group and ϕ, ψ be group actions of G on X respectively Y.
Then f is ϕ-**invariant** if $f(\phi(g, x)) = f(x)$ for all $x \in X$, $g \in G$.
f is (ϕ, ψ)-**equivariant** if $f(\phi(g, x)) = \psi(g, f(x))$ for all $x \in X$, $g \in G$.

Invariance is the same as (ϕ, id)-equivariance.

General DR-Rules for Symmetry Properties

Definition: Invariance and Equivariance

Let $f : X \to Y$, $(G, *)$ a group and ϕ, ψ be group actions of G on X respectively Y.
Then f is ϕ-**invariant** if $f(\phi(g, x)) = f(x)$ for all $x \in X$, $g \in G$.
f is (ϕ, ψ)-**equivariant** if $f(\phi(g, x)) = \psi(g, f(x))$ for all $x \in X$, $g \in G$.

Invariance is the same as (ϕ, id)-equivariance.

$$\frac{K, \sigma \ \phi\text{-invariant} \qquad d \ \phi\text{-invariant}}{\mathcal{R}(K, d) \ \phi\text{-invariant}} \qquad \frac{K, \sigma \ (\phi, \psi)\text{-equivariant} \qquad d \ \phi\text{-invariant}}{\mathcal{R}(K, d) \ (\phi, \psi)\text{-equivariant}}$$

Figure: General Inference Rules (Hadjibeyli and Wilson, 2016)

Abstract Voting and Symmetries Based on Distances

fun *distance-infimum* :: *'a Distance* ⇒ *'a set* ⇒ *'a* ⇒ *ereal* **where**
distance-infimum d A a = *Inf* (*d a* ‘ *A*)

fun *closest-preimg-distance* :: (*'a* ⇒ *'b*) ⇒ *'a set* ⇒ *'a Distance* ⇒
'a ⇒ *'b* ⇒ *ereal* **where**
closest-preimg-distance f domain$_f$ *d a b* = *distance-infimum d* (*preimg f domain*$_f$ *b*) *a*

fun *minimizer* :: (*'a* ⇒ *'b*) ⇒ *'a set* ⇒ *'a Distance* ⇒ *'b set* ⇒ *'a* ⇒ *'b set* **where**
minimizer f domain$_f$ *d A a* = *arg-min-set* (*closest-preimg-distance f domain*$_f$ *d a*) *A*

Abstract Voting and Symmetries Based on Distances

fun *distance-infimum* :: *'a Distance* ⇒ *'a set* ⇒ *'a* ⇒ *ereal* **where**
  distance-infimum d A a = *Inf* (*d a* ‘ *A*)

fun *closest-preimg-distance* :: (*'a* ⇒ *'b*) ⇒ *'a set* ⇒ *'a Distance* ⇒
      'a ⇒ *'b* ⇒ *ereal* **where**
  closest-preimg-distance f domain$_f$ *d a b* = *distance-infimum d* (*preimg f domain*$_f$ *b*) *a*

fun *minimizer* :: (*'a* ⇒ *'b*) ⇒ *'a set* ⇒ *'a Distance* ⇒ *'b set* ⇒ *'a* ⇒ *'b set* **where**
  minimizer f domain$_f$ *d A a* = *arg-min-set* (*closest-preimg-distance f domain*$_f$ *d a*) *A*

lemma (**in** *result*) $\mathcal{R}_{\mathcal{W}}$*-is-minimizer*:
  fixes
    d :: (*'a*, *'v*) *Election Distance* **and**
    C :: (*'a*, *'v*, *'r Result*) *Consensus-Class*
  shows *fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) =
    (λ *E*. ⋃ (*minimizer* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*
                (*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)) *E*))

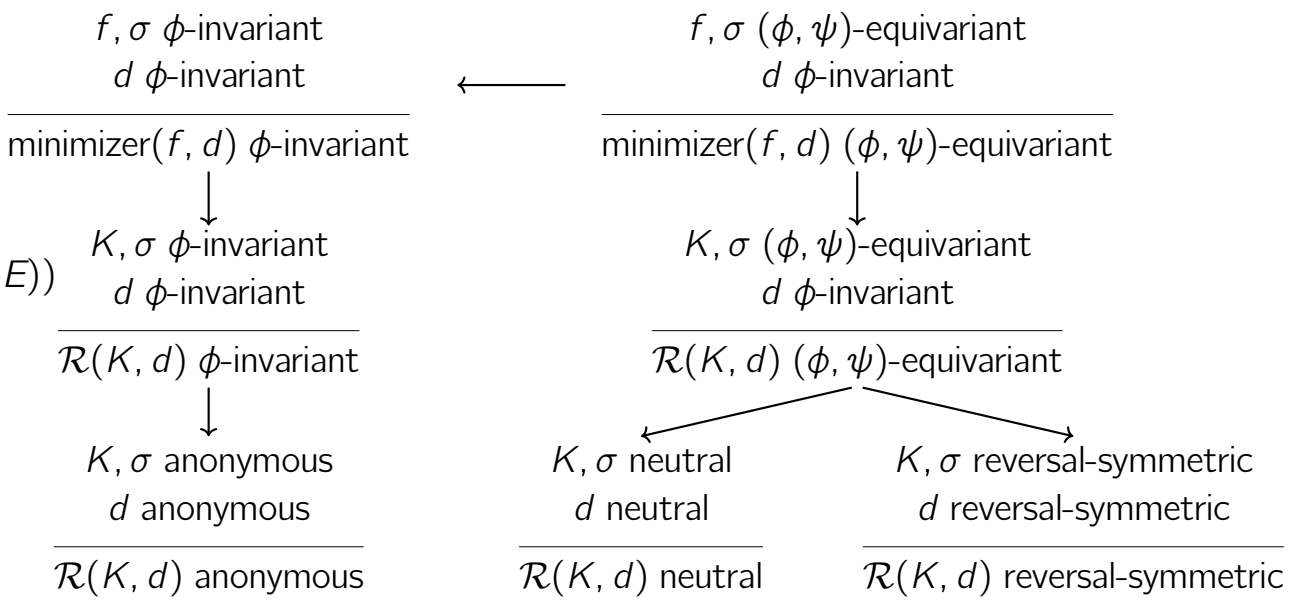
Abstract Voting and Symmetries Based on Distances

fun *distance-infimum* :: *'a Distance* ⇒ *'a set* ⇒ *'a* ⇒ *ereal* **where**
distance-infimum d A a = *Inf* (*d a* ‘ *A*)

fun *closest-preimg-distance* :: (*'a* ⇒ *'b*) ⇒ *'a set* ⇒ *'a Distance* ⇒
'a ⇒ *'b* ⇒ *ereal* **where**
closest-preimg-distance f domain$_f$ *d a b* = *distance-infimum d* (*preimg f domain*$_f$ *b*) *a*

fun *minimizer* :: (*'a* ⇒ *'b*) ⇒ *'a set* ⇒ *'a Distance* ⇒ *'b set* ⇒ *'a* ⇒ *'b set* **where**
minimizer f domain$_f$ *d A a* = *arg-min-set* (*closest-preimg-distance f domain*$_f$ *d a*) *A*

lemma (**in** *result*) $\mathcal{R}_{\mathcal{W}}$*-is-minimizer*:
fixes
d :: (*'a*, *'v*) *Election Distance* **and**
C :: (*'a*, *'v*, *'r Result*) *Consensus-Class*
shows *fun*$_{\mathcal{E}}$ ($\mathcal{R}_{\mathcal{W}}$ *d C*) =
(λ *E*. ⋃ (*minimizer* (*elect-r* ∘ *fun*$_{\mathcal{E}}$ (*rule-*$\mathcal{K}$ *C*)) (*elections-*$\mathcal{K}$ *C*) *d*
(*singleton-set-system* (*limit* (*alternatives-*$\mathcal{E}$ *E*) *UNIV*)) *E*))



Verification of Symmetry Properties

Blueprint for Proof of Symmetry P

(1) Define P as (ϕ, ψ)-equivariance (under a group G)
   (a) Prove that G is a group
   (b) Prove that ϕ is a group action of G on $\mathcal{E}$
   (c) Prove that ψ is a group action of G on UNIV

(2) Prove that σ is (ϕ, ψ)-equivariant

(3) **Add rule for P from general inference rules.**

(4) Prove that K and d have property P (i.e., d is ϕ-invariant)
   (a) Prove that the domain of K is closed under ϕ.
   (b) Prove that K is (ϕ, ψ)-equivariant on its domain.

Verification of Symmetry Properties

Blueprint for Proof of Symmetry P

(1) Define P as (ϕ, ψ)-equivariance (under a group G)
   (a) Prove that G is a group
   (b) Prove that ϕ is a group action of G on $\mathcal{E}$
   (c) Prove that ψ is a group action of G on UNIV

(2) Prove that σ is (ϕ, ψ)-equivariant

(3) **Add rule for P from general inference rules.**

(4) Prove that K and d have property P (i.e., d is ϕ-invariant)
   (a) Prove that the domain of K is closed under ϕ.
   (b) Prove that K is (ϕ, ψ)-equivariant on its domain.

Formalization of Symmetry Properties

locale *result-properties* = *result* +
  fixes ψ :: ('*a* ⇒ '*a*, '*b*) *binary-fun* **and**
    ν :: '*v itself*
  assumes
    action-neutral: *group-action bijection*$_{\mathcal{AG}}$ *UNIV* ψ **and**
    neutrality:
      is-symmetry (λ $\mathcal{E}$ :: ('*a*, '*v*) *Election*. *limit* (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
        (*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$)
          well-formed-elections
          (*φ-neutral well-formed-elections*) (*set-action* ψ))

Verification of Symmetry Properties

Blueprint for Proof of Symmetry P

(1) Define P as (ϕ, ψ)-equivariance (under a group G)
   (a) Prove that G is a group
   (b) Prove that ϕ is a group action of G on $\mathcal{E}$
   (c) Prove that ψ is a group action of G on UNIV

(2) Prove that σ is (ϕ, ψ)-equivariant

(3) **Add rule for P from general inference rules.**

(4) Prove that K and d have property P (i.e., d is ϕ-invariant)
   (a) Prove that the domain of K is closed under ϕ.
   (b) Prove that K is (ϕ, ψ)-equivariant on its domain.

Formalization of Symmetry Properties

locale *result-properties* = *result* +
  fixes ψ :: ('*a* ⇒ '*a*, '*b*) *binary-fun* **and**
    ν :: '*v itself*
  assumes
    action-neutral: *group-action bijection*$_{\mathcal{AG}}$ *UNIV* ψ **and**
    neutrality:
      is-symmetry (λ $\mathcal{E}$ :: ('*a*, '*v*) *Election*. *limit* (*alternatives-*$\mathcal{E}$ $\mathcal{E}$) *UNIV*)
        (*action-induced-equivariance* (*carrier bijection*$_{\mathcal{AG}}$)
          well-formed-elections
          (*φ-neutral well-formed-elections*) (*set-action* ψ))

Property-Specific ("Non-Reusable") Proof Effort

Step (1): homomorphism axiom $(\phi(g, \cdot) \circ \phi(h, \cdot) = \phi(g * h, \cdot))$. Steps (2), (3), (4): Depends on σ and the group action.

Conclusion

Summary

- Devised formal methods for systematic cross-layer development of trustworthy voting systems
- Methods enable reasoning/comprehensibility across system layers by automatic synthesis

Conclusion

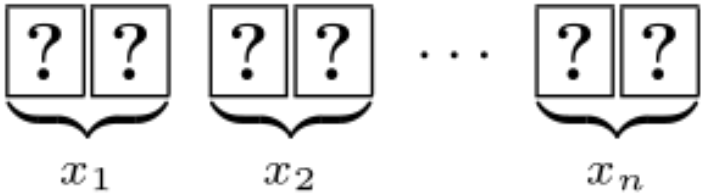
Summary

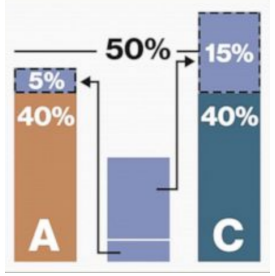
- Devised formal methods for systematic cross-layer development of trustworthy voting systems
- Methods enable reasoning/comprehensibility across system layers by automatic synthesis

Evaluated methods on various **vote tallying algorithms:** Condorcet procedures and classical knock-out tournament

Open Research Questions

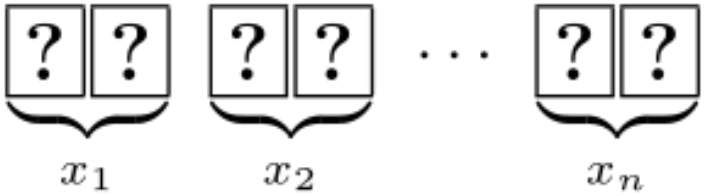
- Extension of methods for vote transfers and committee elections

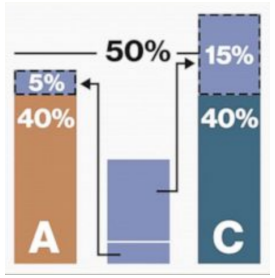




Open Research Questions

- Extension of methods for vote transfers and committee elections

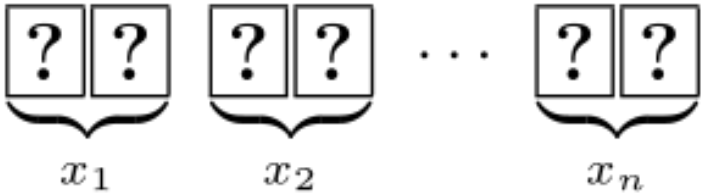


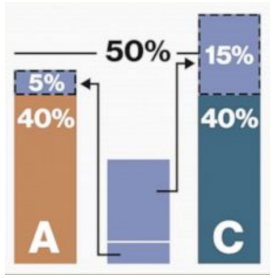


- Flexibilization of robust-by-construction framework for quantitative reasoning

Open Research Questions

- Extension of methods for vote transfers and committee elections



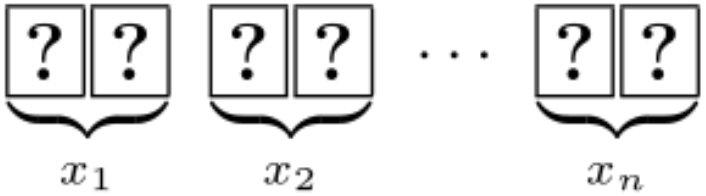


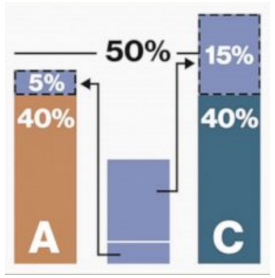
- Flexibilization of robust-by-construction framework for quantitative reasoning

- Integration of cryptographically secure computations into robust-by-construction framework

Open Research Questions

- Extension of methods for vote transfers and committee elections

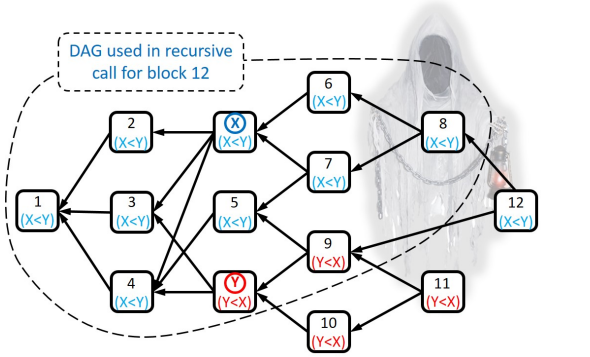




- Flexibilization of robust-by-construction framework for quantitative reasoning

- Integration of cryptographically secure computations into robust-by-construction framework

- Transfer of methods to further domains, e.g., consensus on distributed ledgers



References I

[1] Pietro Grilli di Cortona et al. *Evaluation and optimization of electoral systems*. Society for Industrial and Applied Mathematics, 1999. DOI: 10.1137/1.9780898719819.

[2] Benjamin Hadjibeyli and Mark C. Wilson. "Distance rationalization of social rules". In: *CoRR* abs/1610.01902 (2016). arXiv: 1610.01902. URL: http://arxiv.org/abs/1610.01902.

[3] Ehud Lerer and Shmuel Nitzan. "Some General Results on the Metric Rationalization for Social Decision Rules". In: *Journal of Economic Theory* 37.1 (1985), pp. 191–201. DOI: 10.1016/0022-0531(85)90036-5.