Automated Reasoning for Social Choice Theory

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Tutorial at AAMAS-2023, London, May 2023
http://www.illc.uva.nl/~ulle/teaching/aamas-2023/

Plan for Today

Exciting trend in *computational social choice:* use of *SAT solvers* to *automate* some of our tasks as researchers. *Very cool. But difficult.*

Objective: To enable you to use this approach in your own research.

Main parts of this tutorial:

- Case study: automating the proof of a classical theorem
- Critique and refinement of the basic approach
- Expanding the scope of the approach: focus on explainability

Hands-on: You can reproduce everything you see here directly on your own machine, using the *Jupyter Notebook* provided. *Try it!*

Social Choice Theory

SCT is the study of methods for *collective decision making*, notably *political* decision making by *economic* agents. Such decision making involves, in particular, the *aggregation* of individual *preferences*.

The methodology of SCT ranges from *Philosophy* to *Mathematics*. SCT is traditionally studied in *Economics* and *Political Science* and it is a close cousin of both *decision theory* and *game theory*.

K.J. Arrow, A.K. Sen, and K. Suzumura (eds), *Handbook of Social Choice and Welfare*, Volume 1. North-Holland, 2002.

F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A.D. Procaccia (eds), *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

Why SCT at AAMAS?

A whopping 10% of papers at AAMAS are about social choice. *Why?*

Two explanations:

- *Historical (SCT $\rightarrow$ AI):* Aggregating the individual views of agents in a multiagent system into a collective view is a core task one has to perform when trying to understand or use that system.
- *Modern (AI $\rightarrow$ SCT):* The toolbox of Computer Science and AI has turned out to be extremely useful when it comes to designing and analysing methods for collective decision making (for people).

What we shall do today fits the second explanation.

Voting Rules

Scenario: n *voters* each report a strict ranking over m *alternatives*.

We want to pick a single *winning alternative* (though *ties* are possible) by means of a *voting rule*. Lots of options. Examples:

- *Plurality:* elect the alternative(s) ranked first most often
- *Plurality with runoff:* familiar from French presidential elections
- *Borda:* award $m-k$ points for for getting ranked in the kth position
- *Copeland:* score $=$ won pairwise runoffs $-$ lost pairwise runoffs

Exercise: *Apply these rules to the profile below! (What are n and m?)*

2 Germans:	Beer	$\succ$	Wine	$\succ$	Milk
3 French people:	Wine	$\succ$	Beer	$\succ$	Milk
4 Dutch people:	Milk	$\succ$	Beer	$\succ$	Wine

Which rule is best (or suitable at all) depends on our requirements.

The Problem of Strategic Manipulation

One requirement (or *axiom*) we might want to impose is that we don't want voters to have an incentive to misrepresent their true preferences.

Remember what happened in Florida in 2000 (*stylised*):

49%:	Bush $\succ$ Gore $\succ$ Nader
20%:	Gore $\succ$ Nader $\succ$ Bush
20%:	Gore $\succ$ Bush $\succ$ Nader
11%:	Nader $\succ$ Gore $\succ$ Bush

Under *Plurality*, Bush will win. Nader supporters had an incentive to pretend they prefer Gore. We say: *Plurality is not strategyproof*.

Exercise: *Is there a better voting rule that avoids this problem?*

The Gibbard-Satterthwaite Theorem

By a famous *impossibility theorem*, at the core of both voting theory and mechanism design, the answer to the previous question is: *No!*

Gibbard-Satterthwaite Theorem: *For $m \geqslant 3$ alternatives, no resolute voting rule is strategyproof, surjective, and nondictatorial.*

Meaning of the new concepts mentioned in the theorem:

- *resolute* = the rule always returns a single winner (no ties)
- *surjective* = each alternative can win for *some* way of voting
- *dictatorial* = the top alternative of some fixed voter always wins

Exercise: *Explain why surjectivity and nondictatorship are needed.*

Exercise: *Show that the theorem does not hold for $m = 2$.*

A. Gibbard. Manipulation of Voting Schemes. *Econometrica*, 1973.

M.A. Satterthwaite. Strategy-proofness and Arrow's Conditions. *JET*, 1975.

Proving the Theorem

G-S is a deep result that long seemed elusive:

- People tried and failed to design strategyproof rules for centuries.
- After Arrow's seminal impossibility theorem (for different axioms) a result *à la* G-S seemed to be "in the air".
- It still took two decades to find the right formulation and prove it.
- The original proofs are hard to digest (the original proof of Arrow's impossibility even was wrong—though the theorem itself was fine).

Today the proof of G-S is well understood (see expository paper below). But new results of this kind are still hard to discover and then prove.

K.J. Arrow. *Social Choice and Individual Values*. John Wiley and Sons, 2nd edition, 1963. First edition published in 1951.

U. Endriss. Logic and Social Choice Theory. In A. Gupta and J. van Benthem (eds), *Logic and Philosophy Today*. College Publications, 2011.

Automated Reasoning

Thus: *need much better methodology to reason about social choice!*

Here's again the theorem:

Gibbard-Satterthwaite Theorem: *For $m \geqslant 3$ alternatives, no resolute voting rule is strategyproof, surjective, and nondictatorial.*

Let's try to get a computer to prove it for us! But proving it for *all* $n \geqslant 1$ (voters) and $m \geqslant 3$ (alternatives) is too ambitious for now ...

Exercise: *For which values of n and m is the theorem most surprising?*

Base Case

So let's prove G-S for $n = 2$ voters and $m = 3$ alternatives!

Credo: Even if (*formally*) the full theorem might not follow easily from this 'base case', (*intuitively*) it will then be entirely unsurprising.

Proof Idea

Go through *all voting rules* for $n = 2$ and $m = 3$ and *check* one by one whether they satisfy our requirements. Confirm theorem if none do.

Exercise: *How many (resolute) voting rules do we need to check?*

Better Idea: Logic Encoding

Bad news: there are a total of $m^{(m!^n)} = 3^{36} = 150094635296999121$ resolute voting rules for us to check. *So this won't work.*

Instead, let's try to describe what we need in a compact way . . .

Define a logical language with propositional *variables* $p_{r,x}$ to say that in *profile* r the outcome should include *alternative* x.

Exercise: *Count the variables for $n = 2$ voters and $m = 3$ alternatives!*

Every *assignments of truth values* to such variables corresponds to a *function from profiles to sets of alternatives*, i.e., a *voting rule*.

Exercise: *This is almost true, but not quite. Do you see the problem?*

Modelling Voting Rules and Axioms

A voting rule must return *at least one* alternative x for every profile r:

$$\varphi_{\text{at-least-one}} \quad = \quad \bigwedge_{r} \left(\bigvee_{x} \, p_{r,x} \right)$$

We obtain a perfect correspondence between *voting rules* and *models* ($=$ satisfying truth assignments) of this formula. *Nice!*

Can use similar formulas to encode *axioms* of interest. Then:

models satisfying formulas	$\widehat{=}$	voting rules satisfying axioms
unsatisfiability	$\widehat{=}$	impossibility theorem

SAT Solving

Can use a *SAT solver* to check formulas (in *CNF*) for unsatisfiability.

DIMACS format: use *list of lists of positive and negative integers* to represent *set of clauses of positive and negative literals*. Example:

`[[1,-2,3],[4,-1]]` represents $(p_1 \vee \neg p_2 \vee p_3) \wedge (p_4 \vee \neg p_1)$

Need: script to generate such formulas!

A. Biere, M. Heule, H. van Maaren, and T. Walsh (eds), *Handbook of Satisfiability*. IOS Press, 2009.

A. Ignatiev, A. Morgado, and J. Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. SAT-2018.

Preferences and Profiles

Fix an enumeration of voters, alternatives, preferences, profiles. Then represent everything as integers: *voters* from `0` to `n-1`, *alternatives* from `0` to `m-1`, *preferences* from `0` to `m!-1`, *profiles* from `0` to $\mathtt{m!}^{\mathtt{n}}-1$.

Now providing these methods becomes a routine programming task:

- `allVoters()`, `allAlternatives()`, `allProfiles()`
- `voters(c)`, `alternatives(c)`, `profiles(c)` for condition `c`
- `iVariants(i,r1,r2)` — are profiles `r1` and `r2` `i`-variants?
- `prefers(i,x,y,r)` — does voter `i` prefer `x` to `y` in profile `r`?
- `top(i,x,r)` — does voter `i` top-rank `x` in profile `r`?
- `strProf(r)` — return a string representation for profile `r`

Consult the *Jupyter Notebook* to see one way of doing this.

Try it!

(*better try this on the Jupyter Notebook instead*)

```
>>> list(allProfiles())
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
 20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35]

>>> strProf(35)
'(210,210)'

>>> prefers(0,2,1,35)
True

>>> alternatives(lambda x : x!=1)
[0,2]
```

Literals

Want propositional *variable* $p_{r,x}$ to say that in *profile* r the outcome should include *alternative* x. Enumerate them from 1 to `m!`$^{\texttt{n}}$ `* m`:

```
def posLiteral(r, x):
    return r * m + x + 1

def negLiteral(r, x):
    return (-1) * posLiteral(r, x)
```

Easy to reverse-engineer this to get method to pretty-print literals:

```
>>> strLiteral(1)
'(012,012)->0'

>>> strLiteral(-108)
'not (210,210)->2'
```

Encoding the Requirements on Voting Rules

Now we can encode our requirements. Recall our basic formula:

$$\varphi_{\text{at-least-one}} \quad = \quad \bigwedge_{r} \left(\bigvee_{x} \; p_{r,x} \right)$$

Translating this into code is immediate:

```
def cnfAtLeastOne():
    cnf = []
    for r in allProfiles():
        cnf.append([posLiteral(r,x) for x in allAlternatives()])
    return cnf
```

Try it on the Jupyter Notebook:

```
>>> cnfAtLeastOne()
[[1,2,3], [4,5,6], [7,8,9], [10,11,12], ..., [106,107,108]]
```

Resoluteness

Resoluteness says that for any profile r and any distinct alternatives x and y, not both alternatives are in the outcome for that profile.

Note: Can restrict last quantification to $x < y$ (taken as numbers).

$$\varphi_{\text{res}} \quad = \quad \bigwedge_{r} \left(\bigwedge_{x} \left(\bigwedge_{y \,|\, x<y} \neg p_{r,x} \vee \neg p_{r,y} \right)\right)$$

Again, coding this is immediate:

```
def cnfResolute():
    cnf = []
    for r in allProfiles():
        for x in allAlternatives():
            for y in alternatives(lambda y : x < y):
                cnf.append([negLiteral(r,x), negLiteral(r,y)])
    return cnf
```

Remark: For the following axioms, we now can *presuppose resoluteness*.

Strategyproofness

SP says: for any voter i, any (truthful) profile r, any of its i-variants r', any alternative x, any alternative y dispreferred to x by i in r, either y (bad) loses in r (truthful) or x (good) loses in r' (manipulated).

$$\varphi_{\mathsf{sp}} \quad = \quad \bigwedge_{i} \left(\bigwedge_{r} \left(\bigwedge_{r' \in i\text{-}\mathsf{var}(r)} \left(\bigwedge_{x} \left(\bigwedge_{y \mid x \succ_i^r y} \neg p_{r,y} \vee \neg p_{r',x} \right)\right)\right)\right)$$

```
def cnfStrategyProof():
    cnf = []
    for i in allVoters():
        for r1 in allProfiles():
            for r2 in profiles(lambda r2 : iVariants(i,r1,r2)):
                for x in allAlternatives():
                    for y in alternatives(lambda y : prefers(i,x,y,r1)):
                        cnf.append([negLiteral(r1,y), negLiteral(r2,x)])
    return cnf
```

Surjectivity

Surjectivity really is a conjunction of disjunctions of conjunctions: for all alternatives x, there is a profile r where x wins *and all others lose*. Could translate to CNF. But given resoluteness, this is easier:

$$\varphi_{\textsf{sur}} \quad = \quad \bigwedge_{x} \left(\bigvee_{r} p_{r,x} \right)$$

```
def cnfSurjective():
    cnf = []
    for x in allAlternatives():
        cnf.append([posLiteral(r,x) for r in allProfiles()])
    return cnf
```

Nondictatorship

A resolute rule is nondictatorial if for every voter i there is a profile r where top(i) loses (so: some alternative x equal to top(i) loses).

$$\varphi_{\text{nd}} \quad = \quad \bigwedge_i \left(\bigvee_r \left(\bigvee_{x \mid x = \text{top}(i)} \neg p_{r,x} \right) \right)$$

```
def cnfNonDictatorial():
    cnf = []
    for i in allVoters():
        clause = []
        for r in allProfiles():
            for x in alternatives(lambda x : top(i,x,r)):
                clause.append(negLiteral(r,x))
        cnf.append(clause)
    return cnf
```

Running the SAT Solver

We need to determine whether the *master formula* is satisfiable:

$$\varphi_{\text{gs}} \quad = \quad \varphi_{\text{at-least-one}} \;\wedge\; \varphi_{\text{res}} \;\wedge\; \varphi_{\text{sp}} \;\wedge\; \varphi_{\text{sur}} \;\wedge\; \varphi_{\text{nd}}$$

Btw: this is a conjunction of 1,445 clauses (using 108 variables).

The method `solve()` provides access to a SAT solver.

Let's see what happens:

```
>>> cnf = ( cnfAtLeastOne() + cnfResolute() + cnfStrategyProof()
...         + cnfSurjective() + cnfNonDictatorial() )

>>>> len(cnf)
1445

>>> solve(cnf)
'UNSATISFIABLE'
```

So φ_{gs} really is unsatisfiable! Thus: G-S for $n=2$ and $m=3$ is true! ✓

Discussion: *Does this count? Do we believe in computer proofs?*

Computer Proofs

We can proof-read our *Python script* just like we would proof-read a mathematical proof. And we can use multiple *SAT solvers* and check they agree. So we can have some confidence in the result.

Missing Pieces

But some pieces are still missing:

- *Does the theorem generalise to arbitrary $n \geqslant 2$ and $m \geqslant 3$?*

  Intuitively almost obvious, though technically not that easy. Basic idea: *induction* over both n and m

- *Why does the theorem hold?* This proof does not tell us.

  But SAT technology can help here as well: *MUS extraction*

Inductive Proof

Recall the theorem we want to prove:

Gibbard-Satterthwaite Theorem: *For $m \geqslant 3$ alternatives, no resolute voting rule is strategyproof, surjective, and nondictatorial.*

Instead we proved:

Base Case Lemma: *For $n = 2$ voters and $m = 3$ alternatives, no resolute voting rule is strategyproof, surjective, and nondictatorial.*

To complete the proof of G-S we require two further lemmas:

- impossible for $n \geqslant 2$ and $m = 3$ $\Rightarrow$ impossible for $n+1$ and $m = 3$
- impossible for $n \geqslant 2$ and $m = 3$ $\Rightarrow$ impossible for n and any $m > 3$

Proving the lemmas is tricky but doable. We won't do it here though. Proofs can be found in the PhD thesis of Pingzhong Tang (2010).

P. Tang. Computer-aided Theorem Discovery: A New Adventure and its Application to Economic Theory. PhD thesis. HKUST, 2010.

MUS Extraction

A *minimally unsatisfiable subset* of an unsatisfiable set φ of clauses is an unsatisfiable subset of φ all proper subsets of which are satisfiable. An MUS can serve as an *explanation* for the unsatisfiablity observed.

Use `getMUS()` to compute an MUS of our 1,445-clause CNF:

```
>>> mus = getMUS(cnf)
>>> len(mus)
199
```

That's much better . . . but not good enough. :(

Remark: This MUS includes all (three) surjectivity-clauses, all (two) nondictatorship-clauses, and 158 (out of 1,296) SP-clauses.

Weakening the Theorem

Geist and Peters suggest a weakening of G-S that works better:

Theorem (Geist and Peters, 2017) *For $n \geqslant 3$ and $m \geqslant 3$, no resolute voting rule is both strategyproof and majoritarian.*

Here being *majoritarian* means that x is elected whenever a strict majority ranks it at the top. Check Jupyter Notebook for the code.

Exercise: *Explain why this is weaker than G-S (at least for $n \geqslant 3$).*

Exercise: *Explain why the theorem is false for the case of $n = 2$.*

C. Geist and D. Peters. Computer-Aided Methods for Social Choice Theory. In U. Endriss (ed), *Trends in Computational Social Choice*. AI Access, 2017.

Proving the Theorem

Try this also for yourself (with `n = 3`):

```
>>> cnf = ( cnfAtLeastOne() + cnfResolute()
...       + cnfStrategyProof() + cnfMajoritarian() )

>>> len(cnf)
12696

>>> solve(cnf)
'UNSATISFIABLE'

>>> mus = getMUS(cnf)
>>> len(mus)
21
```

So need to inspect just 21 of the 12,696 clauses to obtain a proof.
Seems feasible. *But can we do even better?*

Shuffle!

Note that an MUS need not be minimal in terms of its *cardinality*. Trying a *different solver*—or just *shuffling* the CNF can help!

Re-run this code a few times and you should get an MUS of size 7:

```
>>> shuffle(cnf)
>>> mus = getMUS(cnf)
>>> len(mus)
7
```

Interpreting the MUS

Here's the small MUS we just found:

```
 >>> print(mus)
[[205,206,207],[-205,-152],[-206,-639],[-207,-199],[199],[152],[639]]
```

Easy to write code to help us interpret this:

```
>>> explainCNF(mus)
AtLeastOne: (102,210,021)->0 or (102,210,021)->1 or (102,210,021)->2
StrategyProof: not (102,210,021)->0 or not (102,102,021)->1
StrategyProof: not (102,210,021)->1 or not (102,210,210)->2
StrategyProof: not (102,210,021)->2 or not (012,210,021)->0
Majoritarian: (012,210,021)->0
Majoritarian: (102,102,021)->1
Majoritarian: (102,210,210)->2
```

The impossibility now is obvious! Done. ✓

Exercise: *The MUS does not include any resoluteness-clauses. Why?*

SAT Solving in the Research Process

What if the MUS is too big? What if the inductive proof doesn't work?

Fear not! In research, knowing what to prove before proving it is rare. Rather: the main challenge is often to identify good hypotheses.

SAT solving can be a great tool for this:

- use the SAT oracle to quickly check base cases for dozens (or hundreds) of axiom variations and combinations
- investigate further only the most interesting of those for which you get an UNSAT result (possibly using entirely traditional methods)

Advanced Exercise

To gain some proficiency with the SAT approach to SCT, try solving this challenging but manageable exercise:

> *Find out about the famous Duggan-Schwartz Theorem and prove its base case using the SAT approach.*

For hints on how to get started, consult Homework #5 for the 2020 edition of my Amsterdam course on COMSOC. The theorem itself is covered in the lecture on *Strategic Manipulation in Voting*.

J. Duggan and T. Schwartz. Strategic Manipulation w/o Resoluteness or Shared Beliefs: Gibbard-Satterthwaite Generalized. *Social Choice and Welfare*, 2000.

U. Endriss. Course on Computational Social Choice. ILLC, University of Amsterdam, 2020. Teaching materials available at `bit.ly/comsoc20ams`.

Literature Review

Geist and Peters (2017) also review parts of the literature up to 2017.

Chatterjee and Sen (2014) comment on early contributions regarding the SAT approach to SCT from the perspective of Economics.

C. Geist and D. Peters. Computer-Aided Methods for Social Choice Theory. In U. Endriss (ed), *Trends in Computational Social Choice*. AI Access, 2017.

S. Chatterjee and A. Sen. Automated Reasoning in Social Choice Theory: Some Remarks. *Mathematics in Computer Science*, 2014.

The Original Paper

Tang and Lin (2009) were the first to use the approach and applied it to construct a new proof of Arrow's Impossibility Theorem.

Their focus was on the inductive proof, while their main impact later turned out to be the idea of automating the proof of the base case.

P. Tang and F. Lin. Computer-Aided Proofs of Arrow's and other Impossibility Theorems. *Artificial Intelligence*, 2009.

Ranking Sets of Objects

In the field of 'ranking sets of objects' people formulate axioms for how to extend an agent's preferences from objects to sets of objects.

In my paper with Christian Geist (2011), we applied the approach to proving impossibility theorems in this new domain. Novel ideas:

- *general reduction lemma:* base case implies full theorem for any combination of axioms meeting certain syntactic constraints
- *automatic theorem discovery:* systematic search over 20 axioms

Found 84 impossibility theorems (some classical; some new—ranging from trivial to interesting; one contradicting wrong result in literature).

C. Geist and U. Endriss. Automated Search for Impossibility Theorems in Social Choice Theory: Ranking Sets of Objects. *JAIR*, 2011.

Tournament Solutions

Brandt and Geist (2016) provide in-depth analysis of strategyproofness for irresolute voting rules (and specifically tournament solutions).

Important paper pioneering *advanced encoding techniques* you might use when a naïve encoding (which was sufficient for G-S) won't work.

F. Brandt and C. Geist. Finding Strategyproof Social Choice Functions via SAT Solving. *JAIR*, 2016.

The No-Show Paradox

Moulin (1988) showed that every *Condorcet-consistent* voting rule suffers from the *no-show paradox:* sometimes it's best to abstain!

Brandt et al. (2017) used the SAT approach to find a *minimal profile* exhibiting the no-show paradox. So here the theorem was known, but SAT helped find a simpler and more interesting base case. Novel ideas:

- *incremental proof discovery:* start with weaker claims and use results to guide proof search over restricted range of profiles
- proof by *graphical representation* extracted from MUS

H. Moulin. Condorcet's Principle Implies the No Show Paradox. *JET*, 1988.

F. Brandt, C, Geist, and D. Peters. Optimal Bounds for the No-Show Paradox via SAT Solving. *Mathematical Social Sciences*, 2017.

Multiwinner Voting

Peters (2018) proved an important result for multiwinner voting regarding the incompatibility of *proportionality* and *strategyproofness* (at least for resolute rules and under mild efficiency requirements).

Particularly worth reading for these reasons:

- insightful discussion of how to look for an impossibility result in a new domain, with *axioms of varying strengths* being considered
- complex *inductive proof* over three variables (n, m, k)

Kluiving et al. (2020) discuss the generalisation to irresolute rules.

D. Peters. Proportionality and Strategyproofness in Multiwinner Elections. AAMAS-2018. Important erratum at `bit.ly/prop-sp-18`.

B. Kluiving, A. de Vries, P. Vrijbergen, A. Boixel, and U. Endriss. Analysing Irresolute Multiwinner Voting Rules with Approval Ballots via SAT Solving. ECAI-2020.

Probabilistic Social Choice

Brandl et al. (2018) used SMT solving (*Satisfiability Modulo Theories*) to obtain results in the domain of probabilistic social choice.

Shows that the approach can work also in domains that might at first seem ill-suited to logic-based approach (due to involving numbers).

F. Brandl, F. Brandt, M. Eberl, and C. Geist. Proving the Incompatibility of Efficiency and Strategyproofness via SMT Solving. *Journal of the ACM*, 2018.

Matching Markets

My 2020 paper applies the SAT approach to matching mechanisms. Includes a particularly simple case of a *general reduction lemma*.

U. Endriss. Analysis of One-to-One Matching Mechanisms via SAT Solving: Impossibilities for Universal Axioms. AAAI-2020.

Fair Division

Brandl et al. (2021) used the SAT approach to settle a 15-year-old conjecture on the impossibility of designing efficient and stratgeyproof rules for distributing money between projects approved by voters.

Aside: Shows that sometimes proofs of claims about models involving numbers might not actually need to refer to those numbers.

F. Brandl, F. Brandt, D. Peters, and C. Stricker. Distribution Rules Under Dichotomous Preferences: Two Out of Three Ain't Bad. EC-2021.

Logic and Social Choice

The SAT approach requires us to model social choice scenarios in logic. The fact that this works so well for the simplest of all logics (propositional logic) actually is somewhat surprising.

Exercise: *What's the main reason why propositional logic was enough?*

Let's briefly discuss related work on logical modelling for SCT.

U. Endriss. Logic and Social Choice Theory. In A. Gupta and J. van Benthem (eds), *Logic and Philosophy Today*. College Publications, 2011.

Logical Minimalism

Why model social choice problems in logic? Besides offering a deeper understanding of SCT and besides sometimes being of direct practical use, there also are philosophical arguments for doing so.

Pauly (2008) argues for *formal minimalism:*

> When considering an axiom in SCT, besides its *normative appeal* and its *logical strength*, we should also take into account the *expressivity of the language* needed to define it. *Less is better*.

This perspective allows us, for instance, to investigate whether a given rule *can be axiomatised* at all, given constraints on language.

M. Pauly. On the Role of Language in Social Choice Theory. *Synthese*, 2008.

Modal Logic

One research direction has been to design tailor-made (usually modal) logics for talking about social choice scenarios.

The paper by Troquard et al. (2011) is a good example (covering G-S).

My paper with Giovanni Ciná (2016) shows how to write out a full *Hilbert-style derivation* of Arrow's Theorem in such a modal logic. It shows that this is possible in principle, but more work is needed to turn this into a tool for automated proof verification or discovery.

N. Troquard, W. van der Hoek, and M. Wooldridge. Reasoning about Social Choice Functions. *Journal of Philosophical Logic*, 2011.

G. Ciná and U. Endriss. Proving Classical Theorems of Social Choice Theory in Modal Logic. *Journal of Autonomous Agents and Multiagent Systems*, 2016.

First-Order Logic

Modelling the G-S conditions in propositional logic worked only due to the restriction to $n = 2$ voters and $m = 3$ alternatives.

In my paper with Umberto Grandi (2013) we explored how close we can get to fully modelling a similar result (Arrow's Theorem) in FOL.

We also document our—largely unsuccessful—attempts to employ *first-order theorem provers* to get a proof. *But others might do better!*

U. Grandi and U. Endriss. First-Order Logic Formalisation of Impossibility Theorems in Preference Aggregation. *Journal of Philosophical Logic*, 2013.

Higher-Order Logic Proof Assistants

There also has been work on verifying the correctness of known proofs of results in SCT using HOL proof assistants such as *Isabelle* or *Coq*.

Nipkow's 2009 paper on Arrow's Theorem and G-S is an example.

T. Nipkow. Social Choice Theory in HOL. *Journal of Automated Reasoning*, 2009.

Formal Verification

A further logic-based application is the use of *model checking* to verify the correctness of *implementations* (e.g., in Java) of voting rules.

Beckert et al. (2017) give an introduction to this topic.

B. Beckert, T. Bormer, R. Goré, M. Kirsten, and C. Schürmann. An Introduction to Voting Rule Verification. In *Trends in COMSOC*. AI Access, 2017.

Beyond Impossibilities

Back to SAT. What are opportunities beyond proving impossibilities?

- proving axioms involved in an impossibility to be *independent*, by showing that every proper subset is satisfiable
- finding an aggregation rule that *satisfies* a given set of axioms
- proving that axioms in Φ *imply* axiom φ: $\textsc{unsat}(\Phi \cup \{\neg\varphi\})$
- *justifying* and *explaining* collective decisions ($\hookrightarrow$ more soon)

However, there are certain challenges associated with some of this:

- might require new ideas to generalise beyond fixed n/m
- interpreting findings saying that CNF is *satisfiable* might be hard

Remark: A very different application of SAT solvers would be to use them to implement computationally intractable aggregation rules.

Example

For $n = 2$ and $m = 3$, how many resolute rules are strategyproof?

This is a question we can answer with the help of our program.

First, construct the corresponding CNF:

```
>>> cnf = cnfAtLeastOne() + cnfResolute() + cnfStrategyProof()
```

We can use `solve()` to find *one* rule satisfying our requirements:

```
>>> solve(cnf)
[1, -2, -3, 4, -5, -6, 7, -8, -9, ..., 106, -107, -108]
```

Exercise: *Write a method to make rule specs such as this readable.*

Using `enumModels()`, we can get *all* such rules (and count them):

```
>>> rules = enumModels(cnf)
>>> len(list(rules))
17
```

Exercise: *What are those 17 rules? Provide a suitable classification.*

Explainability in Social Choice

How do you explain why a given collective decision is the right one?

The axiomatic method seems relevant, given that axioms can motivate voting rules, which in turn produce decisions when applied to profiles.

$$\begin{tikzcd}\text{axioms} \arrow[r] \arrow[rr, dashed, bend right, "?"'] & \text{rules} \arrow[r] & \text{decisions}\end{tikzcd}$$

Example

■ $\succ$ ▲ $\succ$ ●

● $\succ$ ▲ $\succ$ ■

■ $\succ$ ▲ $\succ$ ●

Exercise: *Can you think of a voting rule that makes* ■ *win?*

Example

■ $\succ$ ▲ $\succ$ ●

● $\succ$ ▲ $\succ$ ■

■ $\succ$ ▲ $\succ$ ●

Exercise: *Can you think of a voting rule that makes* ▲ *win?*

Example

■ $\succ$ ▲ $\succ$ ●

● $\succ$ ▲ $\succ$ ■

■ $\succ$ ▲ $\succ$ ●

What's a good outcome?
Why?

Example

■ ≻ ▲ ≻ ● $\mapsto$ {■}

Clear winner!

(FAITHFULNESS)

● ≻ ▲ ≻ ■

■ ≻ ▲ ≻ ●

Example

■ $\succ$ ▲ $\succ$ ● $\mapsto$ {■}

Clear winner!

(FAITHFULNESS)

● $\succ$ ▲ $\succ$ ■

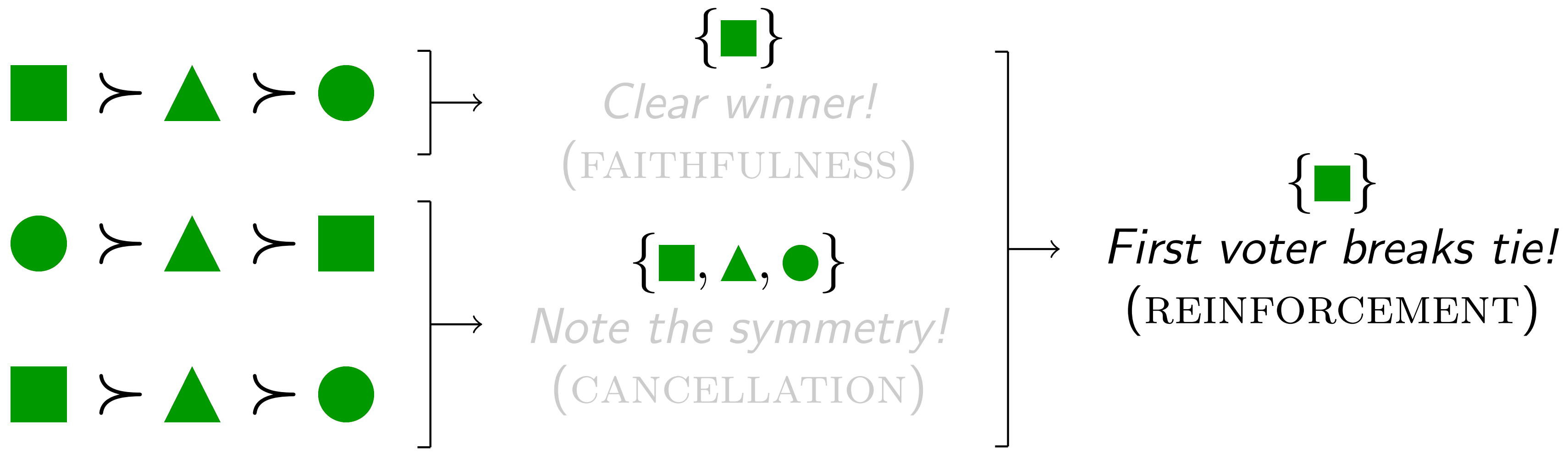
■ $\succ$ ▲ $\succ$ ●

$\mapsto$ {■, ▲, ●}

Note the symmetry!

(CANCELLATION)

Example


■ ≻ ▲ ≻ ●
● ≻ ▲ ≻ ■
■ ≻ ▲ ≻ ●
{■}
Clear winner!
(FAITHFULNESS)
{■, ▲, ●}
Note the symmetry!
(CANCELLATION)
{■}
First voter breaks tie!
(REINFORCEMENT)

Justification = Normative Basis + Explanation

How do you justify selecting outcome $X^\star$ for a given preference profile?

Find axiom set $\mathcal{A}^{\mathrm{NB}}$ (*normative basis*) and set of axiom instances $\mathcal{A}^{\mathrm{EX}}$ (*explanation*) regarding specific scenarios meeting these conditions:

- *Adequacy:* axioms in $\mathcal{A}^{\mathrm{NB}}$ are acceptable to the user
- *Relevance:* $\mathcal{A}^{\mathrm{EX}}$ only includes instances of axioms in $\mathcal{A}^{\mathrm{NB}}$
- *Explanatoriness:* every voting rule satisfying $\mathcal{A}^{\mathrm{EX}}$ returns $X^\star$ (and $\mathcal{A}^{\mathrm{EX}}$ is tight: none of its proper subsets have the same property)
- *Nontriviality:* at least one voting rule satisfies $\mathcal{A}^{\mathrm{NB}}$

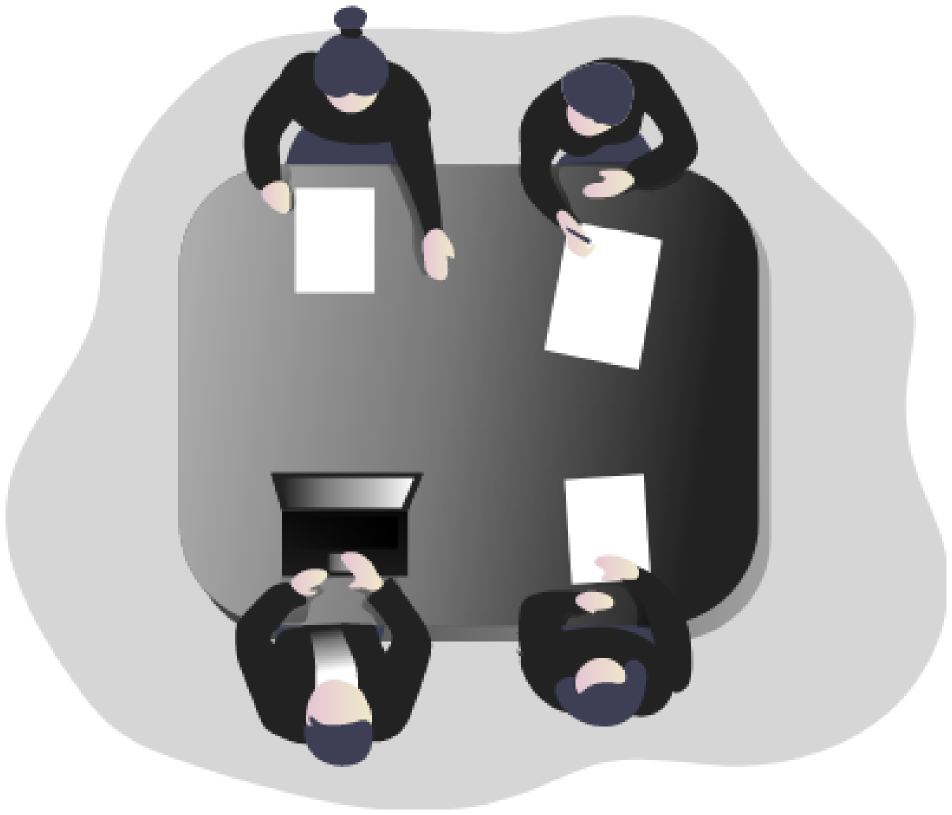
We can operationalise all of this using SAT-solving technology!

Main idea is to compute MUS of all instances of all acceptable axioms, together with formula saying that $X^\star$ is *not* selected in given profile.

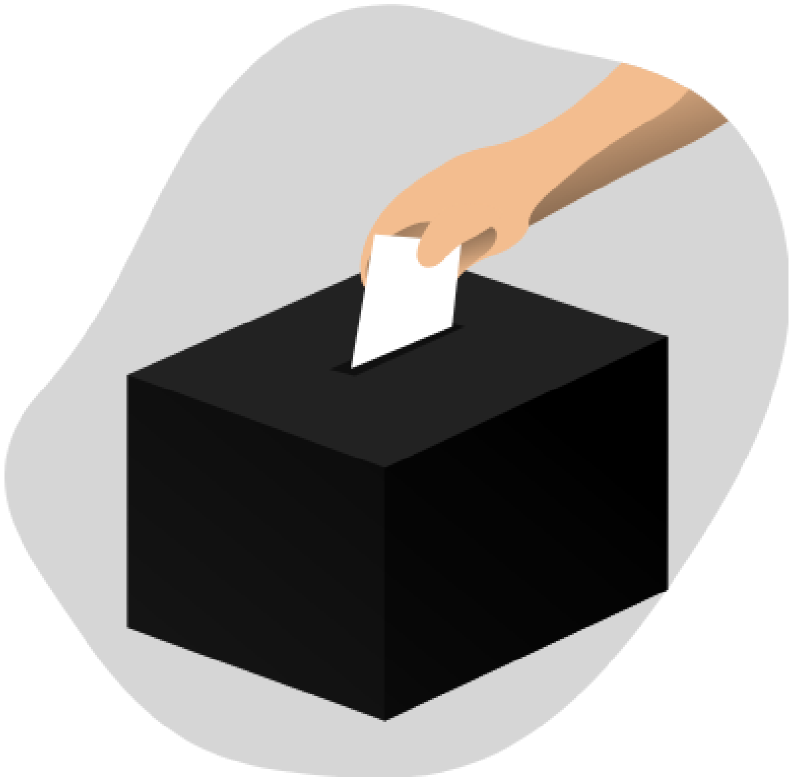
A. Boixel and U. Endriss. Automated Justification of Collective Decisions via Constraint Solving. AAMAS-2020.

Scenario 1: Confidence in Election Results

Scenario 2: Deliberation Support



Scenario 3: Justification Generation as Voting



M.C. Schmidtlein and U. Endriss. Voting by Axioms. AAMAS-2023.

Demo

Use this tool to compute an axiomatic justification and a step-by-step explanation for a preference profile and target outcome of your choice:

`bit.ly/xsoc-demo`

A. Boixel, U. Endriss, and O. Nardi. Displaying Justifications for Collective Decisions. IJCAI-2022 Demo Track.

Broader Perspective

Consult my paper with Olivier Cailloux (2016), the position paper by Procaccia (2019), and the survey by Suryanarayana et al. (2022) for a broader discussion of explainability in multiagent decision making.

O. Cailloux and U. Endriss. Arguing about Voting Rules. AAMAS-2016.

A.D. Procaccia. Axioms Should Explain Solutions. In J.-F. Laslier et al. (eds), *The Future of Economic Design*. Springer, 2019.

S.A. Suryanarayana, D. Sarne, and S. Kraus. Explainability in Mechanism Design: Recent Advances and the Road Ahead. EUMAS-2022.

Beyond SAT

Beyond SAT, there are also a number of other (often logic-based) tools one might try using in similar ways as we used SAT solvers:

- (Mixed) Integer Programming
- Constraint Programming
- SAT Modulo Theories (SMT)
- Answer Set Programming
- First-Order Theorem Proving

Last Slide

This has been an introduction to the use of automated reasoning, and specifically SAT solving, in social choice theory.

- Impossibilities: base case via SAT, human-readable proof via MUS, full proof via induction (*or just use the approach as a heuristc!*)
- Beyond impossibilities: explainability (+ *much more*)

Lots of opportunities still to be explored!