Computational Social Choice: Spring 2019

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

Plan for Today

Obtaining axiomatic results in SCT is hard: eliminating various minor errors from the original proof of Arrow's Theorem took several years; the Gibbard-Satterthwaite Theorem was conjectured at least a decade before it was proved correct; getting new results is really challenging.

Can *automated reasoning*, as studied in AI, help? *Yes!*

Today we focus on one such approach (case study: G-S Theorem):

- encode a social choice scenario into *propositional logic*
- reason about this encoding with the help of a *SAT solver*

Consult Geist and Peters (2017) for an introduction to this approach.

But first: general remarks on *logic and automated reasoning* for SCT

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

Logic for Social Choice Theory

It can be insightful to model SCT problems in logic (Pauly, 2008):

- One research direction is to explore how far we can get using a *standard logic*, such as classical FOL. Do we need second-order constructs to capture IIA? (Grandi and Endriss, 2013)
- Another direction is to design *tailor-made logics* specifically for SCT (for instance, a modal logic). Can we cast the proof of Arrow's Theorem in natural deduction? (Ciná and Endriss, 2016)

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

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

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

Automated Reasoning Approaches

Logic has long been used to help verify the correctness of hardware and software. Can we use this methodology also here? *Yes!*

- Automated *verification* of a (known) proof of Arrow's Theorem in the HOL proof assistant ISABELLE (Nipkow, 2009).
- Automated *proof* of Arrow's Theorem for 3 alternatives and 2 voters using a SAT solver (Tang and Lin, 2009).
- Use of *model checking* to verify correctness of *implementations* (e.g., in Java) of voting rules (Beckert et al., 2017).

We will now focus on the second approach above.

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

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

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

Case Study: The Gibbard-Satterthwaite Theorem

Recall this central theorem of social choice theory:

Theorem 1 (Gibbard-Satterthwaite) *There exists no resolute SCF for $\geqslant 3$ alternatives that is surjective, strategyproof, and nondictatorial.*

Remark: The theorem is trivially true for $n = 1$ voter. (*Why?*)

We will now discuss an alternative proof:

- We use a *SAT solver* to automatically prove that the theorem holds for the *smallest nontrivial case* (with $n = 2$ and $m = 3$).
  Note: $3^{3! \times 3!} \approx 1.5 \times 10^{17}$ functions to check!
- Using purely theoretical means, we prove that this entails the theorem for *all larger values of n and m* (as long as n is finite).

A. Gibbard. Manipul. of Voting Schemes: A General Result. *Econometrica*, 1973.

M.A. Satterthwaite. Strategy-proofness and Arrow's Conditions. *Journal of Economic Theory*, 1975.

Approach

We will use *Lingeling*, a SAT solver developed by the formal methods group at Johannes Kepler University Linz (`fmv.jku.at/lingeling/`).

Lingeling can check whether a given formula in CNF is satisfiable. The formula must be represented as a *list of lists of integers*, corresponding to a *conjunction of disjunctions of literals*. Positive (negative) numbers represent positive (negative) literals.

Example: `[[1,-2,3], [-1,4]]` represents $(p \vee \neg q \vee r) \wedge (\neg p \vee s)$.

We will use a *Python* script (Python3) to generate a propositional formula $\varphi_{\textsf{GS}}$ that is satisfiable iff there exists a resolute SCF for $n = 2$ voters and $m = 3$ alternatives that is surjective, SP, and nondictatorial. Using Lingeling, we will show that $\varphi_{\textsf{GS}}$ *is not satisfiable*.

To access Lingeling from Python we will use the library `pylgl`, which provides a function `solve` (`pypi.org/project/pylgl/`).

Example: `solve([[1], [-1,2], [-2]])` will result in `'UNSAT'`. ✓

Basics: Voters, Alternatives, Profiles

We first fix n (*number of voters*) and m (*number of alternatives*):

```
n = 2
m = 3
```

Voters and alternatives are referred to by number (starting from `0`). Functions to retrieve the *lists of all voters* and *all alternatives:*

```
def allVoters():
    return range(n)

def allAlternatives():
    return range(m)
```

There are $(m!)^n$ different profiles. We refer to them by number as well. Function to retrieve the *list of all profiles:*

```
from math import factorial

def allProfiles():
    return range(factorial(m) ** n)
```

Working with Permutations

We will model *preferences as permutations* of the set of alternatives. The most complicated piece of code we need is a function to return the *nth permutation of a given list L* (with $n \in \{0, \ldots, |L|! - 1\}$):

```
def nthPerm(num, mylist):
    length = len(mylist)
    if length > 1:
        pos = num // factorial(length-1)
        restnum = num - pos * factorial(length-1)
        restlist = mylist[:pos] + mylist[pos+1:]
        return [mylist[pos]] + nthPerm(restnum, restlist)
    else:
        return [mylist[0]]
```

This works as intended:

```
>>> nthPerm(1, [0,1,2])          >>> nthPerm(5, [0,1,2])
[0, 2, 1]                        [2, 1, 0]
```

Extracting Preferences

Also preferences are referred to by number (between 0 and $m! - 1$).
Function to return the *preference of voter i in profile $\boldsymbol{R}$:*

```
def preference(i, r):
    fact = factorial(m)
    return ( r % (fact ** (i+1)) ) // (fact ** i)
```

Think of profile numbers as *$m!$-ary numbers* (digits $=$ preferences).

Example: For $n = 5$ and $m = 3$, to extract the preference of voter 1 in profile number 99, note that $99 = 0 \cdot 6^4 + 0 \cdot 6^3 + 2 \cdot 6^2 + 4 \cdot 6^1 + 3 \cdot 6^0$.
So her preference order is the 4th permutation of `[0,1,2]`.

Interpreting Preferences

Putting together our functions for extracting (numbers representing) preferences from a given (number representing a) profile and for handling permutations, it is now straightforward to provide a function to check whether *voter i prefers alternative x to y in profile $\boldsymbol{R}$:*

```
def prefers(i, x, y, r):
    mylist =  nthPerm(preference(i,r), list(allAlternatives()))
    return mylist.index(x) < mylist.index(y)
```

Function to check whether *x is voter i's top alternative in profile $\boldsymbol{R}$:*

```
def top(i, x, r):
    mylist =  nthPerm(preference(i,r), list(allAlternatives()))
    return mylist.index(x) == 0
```

Restricting the Range of Quantification

When formulating axioms, we sometimes need to quantify over all alternatives that satisfy a certain (boolean) condition:

```
def alternatives(condition):
    return [x for x in allAlternatives() if condition(x)]
```

Example: You can now generate the list of all alternatives that meet the condition of being different from 1 (`condition` $= \lambda x.(x \neq 1)$).

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

And the corresponding functions for voters and profiles:

```
def voters(condition):
    return [i for i in allVoters() if condition(i)]

def profiles(condition):
    return [r for r in allProfiles() if condition(r)]
```

Literals

We can specify any (possibly irresolute) SCF $F : \mathcal{L}(A)^n \to 2^A \setminus \{\emptyset\}$ by saying whether or not $x \in F(\boldsymbol{R})$ for every profile $\boldsymbol{R}$ and alternative x.

So create a propositional variable $p_{\boldsymbol{R},x}$ for every profile $\boldsymbol{R} \in \mathcal{L}(A)^n$ and every alternative $x \in A$, with the intended meaning that:

$$p_{\boldsymbol{R},x} \text{ is } \textit{true} \ \underline{\text{iff}} \ x \in F(\boldsymbol{R})$$

Exercise: *How many variables for $n = 2$ voters and $m = 3$ alternatives?*

Need to decide *which number* to use to represent $p_{\boldsymbol{R},x}$. Good option:

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

Recall: `r` $\in \{0, \ldots, (m!)^n - 1\}$
and `x` $\in \{0, \ldots, m-1\}$

And negative literals are represented by negative numbers:

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

Modelling Social Choice Functions

Every assignment of truth values to our *108 variables* $p_{\boldsymbol{R},x}$ corresponds to a function $F : \mathcal{L}(A)^n \to 2^A$ (in case $n = 2$ and $|A| = 3$).

But: a (possibly irresolute) SCF is a function $F : \mathcal{L}(A)^n \to 2^A \setminus \{\emptyset\}$.

Fix this by restricting attention to models of this formula:

$$\varphi_{\text{at-least-one}} \quad = \quad \bigwedge_{\boldsymbol{R} \in \mathcal{L}(A)^n} \left(\bigvee_{x \in A} p_{\boldsymbol{R},x} \right)$$

The following function will generate this formula:

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

At Least One Winning Alternative

Let's give it a try:

```
>>> cnfAtLeastOne()
[[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, 36], [37, 38, 39], [40, 41,
42], [43, 44, 45], [46, 47, 48], [49, 50, 51], [52, 53, 54],
[55, 56, 57], [58, 59, 60], [61, 62, 63], [64, 65, 66], [67,
68, 69], [70, 71, 72], [73, 74, 75], [76, 77, 78], [79, 80,
81], [82, 83, 84], [85, 86, 87], [88, 89, 90], [91, 92, 93],
[94, 95, 96], [97, 98, 99], [100, 101, 102], [103, 104, 105],
[106, 107, 108]]
```

Nice: we really get $(3!)^2 = 36$ clauses of 3 positive literals each.

Resoluteness

We now write similar functions for each one of our axioms.

F is *resolute* if for *all* profiles $\boldsymbol{R}$ and *all* alternatives x and y it is the case that $x \notin F(\boldsymbol{R})$ *or* $y \notin F(\boldsymbol{R})$. So: *at most one winner* per profile.

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

$$\varphi_{\text{resolute}} \quad = \quad \bigwedge_{\boldsymbol{R} \in \mathcal{L}(A)^n} \left(\bigwedge_{x \in A} \left(\bigwedge_{\substack{y \in A \\ \text{s.t. } x < y}} \neg p_{\boldsymbol{R},x} \vee \neg p_{\boldsymbol{R},y} \right) \right)$$

```
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
```

Surjectivity

Surjectivity is most naturally expressed as a conjunction of disjunctions of conjunctions (*how?*). Could translate to CNF, but this is easier:

If F is already known to be *resolute*, then F is *surjective* if for *all* alternatives x there *exists* a profile $\boldsymbol{R}$ such that $x \in F(\boldsymbol{R})$.

$$\varphi_{\textsf{surjective}} \quad = \quad \bigwedge_{x \in A} \left(\bigvee_{\boldsymbol{R} \in \mathcal{L}(A)^n} p_{\boldsymbol{R},x} \right)$$

So: every alternative is amongst the winners in at least one profile.

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

Preparation for Modelling Strategyproofness

To model strategyproofness we need to be able to model that two profiles are so-called *i-variants* (for some agent $i \in N$):

$$\boldsymbol{R} =_{-i} \boldsymbol{R}' \ \ \underline{\text{iff}} \ \ R_j = R'_j \text{ for all agents } j \in N \setminus \{i\}$$

Recall: `preference(j,r)` returns the preference of voter `j` in profile `r`

Now our implementation is straightforward:

```
def iVariants(i, r1, r2):
    return all(preference(j,r1) ==
                  preference(j,r2) for j in voters(lambda j : j!=i))
```

Strategyproofness

Resolute F is *strategyproof* if for *all* voters i, *all* (truthful) profiles $\boldsymbol{R}_1$, *all* of its *i-variants* $\boldsymbol{R}_2$, *all* alternatives x, and *all* alternatives y *dispreferred* to x by i in $\boldsymbol{R}_1$ we have: $F(\boldsymbol{R}_1) = y$ *implies* $F(\boldsymbol{R}_2) \neq x$.

$$\varphi_{\mathsf{SP}} = \bigwedge_{i \in N} \left(\bigwedge_{\boldsymbol{R}_1 \in \mathcal{L}(A)^n} \left(\bigwedge_{\substack{\boldsymbol{R}_2 \in \mathcal{L}(A)^n \\ \text{s.t. } \boldsymbol{R}_1 =_{-i} \boldsymbol{R}_2}} \left(\bigwedge_{x \in A} \left(\bigwedge_{\substack{y \in A \\ \text{s.t. } i \in N^{\boldsymbol{R}_1}_{x \succ y}}} \neg p_{\boldsymbol{R}_1, y} \vee \neg p_{\boldsymbol{R}_2, x} \right)\right)\right)\right)$$

```
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
```

Nondictatorship

Resolute F is *nondictatorial* if for *all* voters i there *exists* a profile $\boldsymbol{R}$ such that $F(\boldsymbol{R}) \neq x$ for alternative $x = \mathrm{top}_i(\boldsymbol{R})$.

$$\varphi_{\text{nondictatorial}} = \bigwedge_{i \in N} \left(\bigvee_{\boldsymbol{R} \in \mathcal{L}(A)^n} \left(\bigvee_{\substack{x \in A \\ \text{s.t. } x = \mathrm{top}_i(\boldsymbol{R})}} \neg p_{\boldsymbol{R},x} \right) \right)$$

this works as $x = \mathrm{top}_i(\boldsymbol{R})$ for just <u>one</u> x

```
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
```

Proving the (Special Case of the) Theorem

Putting it all together:

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

This is a conjunction of 1445 clauses (using 108 variables, as we saw):

```
>>> len(cnf)
1445
```

We make Lingeling available like this:

```
from pylgl import solve
```

And now the moment of truth has come:

```
>>> solve(cnf)
'UNSAT'
```

Done! So the G-S Theorem really holds for $n = 2$ and $m = 3$. Nice. ✓

Discussion: Confidence in Computer Proofs?

Some will object to this approach. *Can we trust this kind of proof?*

Your computer-generated proof using a SAT solver is valid only if:

- your *encoding* of your question into propositional logic is correct
- the implementation of the *SAT solver* is correct
- the *environment* the solver is running in works to specification

Fortunately, there are some pretty good counterarguments:

- Correctness of (leading) SAT solvers not an issue: was *scrutinised* by many more people than most manual proofs in the literature.

  So, if the part you implement yourself is short and clean (if it can be printed in a paper submitted for publication), then you are ok.
- Due to standardised input/output format for SAT solvers, you can *verify* the correctness of your proof using *third-party tools*.
- Sometimes you can automatically extract a *human-readable proof*.

Completing the Proof of the G-S Theorem

We now have a proof of the Gibbard-Satterthwaite Theorem for the *special case* of $n = 2$ voters and $m = 3$ alternatives. Next we show:

- 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 $m > 3$

Observe how this entails an impossibility result for *all* $n \geqslant 2$ and $m \geqslant 3$.

Next: Proofs of (the contrapositives of) the above two lemmas.

Remark: Recall that we had seen in an earlier lecture that any resolute SCF that is both *surjective* and *strategyproof* must also be *Paretian*. We will use this fact for the proofs of both lemmas.

First Lemma

Lemma 2 *If there exists a resolute SCF for $n+1>2$ voters and three alternatives that is surjective, strategyproof, and nondictatorial, then there also exists such a SCF for n voters and three alternatives.*

Proof: Let $A=\{a,b,c\}$ and $N=\{1,\ldots,n\}$. Now take any resolute SCF $F:\mathcal{L}(A)^{n+1}\to A$ that is surjective, SP, and nondictatorial.

For every $i\in N$, define $F_i:\mathcal{L}(A)^n\to A$ via $F_i(\boldsymbol{R})=F(\boldsymbol{R},R_i)$. And check:

- All F_i are *surjective:* Immediate from F being Paretian. ✓
- All F_i are *SP:* First, no $j\neq i$ can manipulate, given that F is SP.

  Now suppose voter i can manipulate in $\boldsymbol{R}$ using R_i'. Thus, i prefers $F(\boldsymbol{R}_{-i},R_i',R_i')$ to $F(\boldsymbol{R}_{-i},R_i,R_i)$. But then i also must prefer $F(\boldsymbol{R}_{-i},R_i',R_i')$ to $F(\boldsymbol{R}_{-i},R_i',R_i)$ or $F(\boldsymbol{R}_{-i},R_i',R_i)$ to $F(\boldsymbol{R}_{-i},R_i,R_i)$. So F is manipulable in both cases. Contradiction. ✓
- At least one F_i is *nondictatorial:* If all F_i are dictatorial, F must elect the top-choice of voter $n{+}1$ whenever at least one other voter submits the same ballot. But any such F is manipulable. Contradiction. ✓

Second Lemma

Lemma 3 *If there exists a resolute SCF for n voters and $m > 3$ alternatives that is surjective, strategyproof, and nondictatorial, then there also exists such a SCF for n voters and three alternatives.*

Proof: Let $m > 3$ and let $A = \{a,b,c,a_4,\ldots,a_m\}$. Take any resolute SCF $F : \mathcal{L}(A)^n \to A$ that is surjective, SP, and nondictatorial.

For any $R \in \mathcal{L}(\{a,b,c\})$, let $R^+ = R(1) \succ R(2) \succ R(3) \succ a_4 \succ \cdots \succ a_m$.

Now define a SCF $F' : \mathcal{L}(\{a,b,c\})^n \to \{a,b,c\}$ for three alternatives:

$$F'(R_1,\ldots,R_n) \quad = \quad F(R_1^+,\ldots,R_n^+)$$

F' is well-defined (really maps to $\{a,b,c\}$) and surjective, because F is Paretian. F' also is immediately seen to be SP and nondictatorial. ✓

Critique of the Approach

A possible objection to this approach is that proving the lemmas can be quite difficult, almost as difficult as proving the theorem itself.

This is a valid concern. But:

- A successful proof for a special case with small n and m provides *strong evidence* for (though no formal proof of) a general result.

  Indeed: The G-S Theorem is surprising. Our lemmas are not at all!

  Can use this as a *heuristic* to decide what to investigate further.

- Sometimes it may be possible to prove a *general lemma:* as long as the axioms involved meet certain conditions, every impossibility established for a small scenario will generalise to all larger ones.

Refinements of the Approach (1)

The approach was originally developed by Tang and Lin (2009) who used it to reprove Arrow's Theorem and related impossibility results.

Since then there have been several refinements of the basic approach:

- General *reduction lemma* that applies to all axioms meeting certain syntactic restrictions. Allows for automated *discovery* (not just verification) of results (Geist and Endriss, 2011).

  Trivia: Christian Geist's MoL thesis from 2010 lists 84 impossibility theorems discovered in a space of 20 axioms (by trying all subsets).

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

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

Refinements of the Approach (2)

Further developments have taken place very recently:

- More *sophisticated encodings* into SAT (Brandt and Geist, 2016).
- Extraction of *minimally unsatisfiable sets* to enable construction of *human-readable proofs* (Brandt and Geist, 2016).
- Extension to SMT solving (*satisfiability modulo theories*). Used to obtain results for probabilistic social choice (Brandl et al., 2018).

F. Brandt and C. Geist. Finding Strategyproof Social Choice Functions via SAT Solving. *Journal of Artificial Intelligence Research*, 2016.

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

Summary

This has been an introduction to the application of tools from logic and automated reasoning to the study of social choice.

Our focus has been on a hands-on example: proving the "base case" of the Gibbard-Satterthwaite Theorem with a SAT solver.

An approach with lots of potential (but steep learning curve!).

Related work discussed only very briefly:

- *logical modelling* of social choice scenarios using a variety of logics
- verification of known proofs using *interactive theorem provers*
- *formal verification of implementations* of voting rules

What next? Applications of knowledge representation techniques.