# Advanced Topics in Computational Social Choice

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

## Plan for Today

Recall how we reduced the base case of the Gibbard-Satterthwaite Theorem to a query to a SAT solver, which confirmed unsatisfiability.

Today we will (try to) address two remaining shortcomings:

- generalisation beyond $n, m = 2, 3$ via an *inductive argument*
- understanding the proof of the base case by inspecting an *MUS*

# 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 during the regular COMSOC course that any resolute voting rule that is *surjective* and *strategyproof* must also be *Paretian*. We will use this fact for the proofs of both lemmas.

# First Lemma

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

Proof: Let $A = \{a, b, c\}$ and $N = \{1, \ldots, n\}$. Now take any resolute rule $F : \mathcal{L}(A)^{n+1} \to A$ 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:* Assume all $F_i$ are dictatorial. If all $F_i$ have same dictator, $F$ is dictatorial (contradiction!). Otherwise, must have $\mathrm{dict}(F_i) = i$, meaning $F$ elects $\mathrm{top}(R_{n+1})$ whenever someone else submits the same ballot. But then $F$ is manipulable (contradiction!). ✓

# Second Lemma

**Lemma 2** *If there exists a resolute voting rule for $n$ voters and $m > 3$ alternatives that is surjective, strategyproof, and nondictatorial, then there also exists such a rule for $n$ voters and three alternatives.*

Proof: Let $m > 3$ and let $A = \{a, b, c, a_4, \ldots, a_m\}$. Take any resolute rule $F : \mathcal{L}(A)^n \to A$ 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 rule $F^{a,b,c} : \mathcal{L}(\{a,b,c\})^n \to \{a,b,c\}$ for three alternatives:

$$F^{a,b,c}(R_1, \ldots, R_n) \quad = \quad F(R_1^+, \ldots, R_n^+)$$

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

Done if $F^{a,b,c}$ is nondictatorial. If not, consider all $F^{x,y,z}$ for $x, y, z \in A$. Done if one of them is nondictatorial. If *all* are dictatorial, get contradiction: As SP implies *independence*, if $F^{a,b,c}$ has dictator $i$, $i$ is "local dictator" for $\{a, b, c\}$ under $F$. So $F$ has some local dictator for every triple. But these local dictators cannot be distinct voters, so $F$ in fact must be dictatorial. ✓

# Critique of the Approach

Proving such 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 you can prove a *general reduction lemma:* if the axioms meet certain conditions, every impossibility generalises from small to large scenarios (see examples cited below).

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

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

## Human-Readable Proofs

As discussed, if we carefully *proof-read* the code used to generate the CNF and if we *reproduce* our result on several SAT solvers, then we should have high confidence in the *correctness* of our result.

But: We still won't know *why* it holds! Need a human-readable proof.

*So how can we extract such a proof from the work done by the solver?*

# Tools

We will continue to write code in Python and to access the SAT solver Lingeling via PyLGL (see first slide set for details).

In addition we will also use the *PicoMUS* tool, shipped with *PicoSAT* (`fmv.jku.at/picosat/`), which should be compiled with trace support.

On these slides, we shall assume that `picosat` and `picomus` have been installed in a directory called `~/solvers/`.

All the code from this and the first slide set is available in `lect2.py`.

# External Analysis of the CNF

So far we have called the SAT solver directly from within Python.
While this is very convenient, there also are downsides:

- smaller range of tools we can use (from PyLGL: only Lingeling)
- reduction in performance (not an issue so far, but could be)

Instead we could store the CNF generated in a text file.

This example illustrates the *DIMACS format* for such files:

```
p cnf 3 4
-1 -2 3 0
-1 3 0
1 0
-3 -1 0
```

There is one clause per line (the `0` indicates the end of a clause).
The first two numbers indicate the numbers of variables and clauses.

Exercise: *What is the CNF encoded by the above sample file?*

# Storing the CNF

You can use this function to save a given CNF to a text file:

```
def saveCNF(cnf, filename):
    nvars = max([abs(lit) for clause in cnf for lit in clause])
    nclauses = len(cnf)
    file = open(filename, 'w')
    file.write('p␣cnf␣' + str(nvars) + '␣' + str(nclauses) + '\n')
    for clause in cnf:
        file.write('␣'.join([str(lit) for lit in clause]) + '␣0\n')
    file.close()
```

Let's try! The following sequence of commands will generate a text file called `test.dimacs` with the contents shown on the previous slide:

```
>>> mycnf = [ [-1,-2,3], [-1,3], [1], [-3,-1] ]
>>> saveCNF(mycnf, 'test.dimacs')
```

## Let's Try!

We now can check the satisfiability of our CNF either directly from Python (as we used to) or by saving it and then running PicoSAT.

```
$ python3 -i lect2.py
>>> mycnf = [ [-1,-2,3], [-1,3], [1], [-3,-1] ]
>>> solve(mycnf)
'UNSAT'
```

```
$ python3 -i lect2.py
>>> mycnf = [ [-1,-2,3], [-1,3], [1], [-3,-1] ]
>>> saveCNF(mycnf, 'test.dimacs')
>>> exit()
$ ~/solvers/picosat test.dimacs
s UNSATISFIABLE
```

## Minimally Unsatisfiable Subsets

Think of a formula in CNF as a (possibly very large!) set of clauses.

To understand *why* a given set $\Phi$ of clauses is unsatisfiable, it can be helpful to inspect a *minimally unsatisfiable subset* (MUS) of $\Phi$.

For a given unsatisfiable set $\Phi$, any set $\Phi^\star$ is called an MUS of $\Phi$ if:

- $\Phi^\star \subseteq \Phi$,
- $\Phi^\star$ is unsatisfiable, but
- every proper subset of $\Phi^\star$ is satisfiable.

We can use *PicoMUS* to search for an MUS for an unsatisfiable CNF.

# Let's Try!

Recall that `test.dimacs` contains an unsatisfiable CNF with 4 clauses.

Let's run PicoMUS on this file and save the result in `mymus.dimacs`:

```
$ ~/solvers/picomus test.dimacs mymus.dimacs
s UNSATISFIABLE
c [picomus] computed MUS of size 3 out of 4 (75%)
...
```

Files `test.dimacs` (left) and `mymus.dimacs` (right):

```
p cnf 3 4
-1 -2 3 0
-1 3 0
1 0
-3 -1 0
```

```
p cnf 3 3
-1 3 0
1 0
-3 -1 0
```

So the first clause was redundant.

# Back to the Gibbard-Satterthwaite Theorem

Recall that our encoding of the G-S Theorem for $n = 2$ and $m = 3$ consists of an unsatisfiable conjunction of 1445 clauses.

Let's try to compute an MUS:

```
>>> cnf = ( cnfAtLeastOne() + cnfResolute() + cnfSurjective()
...       + cnfStrategyProof() + cnfNondictatorial() )
>>> saveCNF(cnf, 'gs.dimacs')
```

```
$ ~/solvers/picomus gs.dimacs mymus.dimacs
s UNSATISFIABLE
c [picomus] computed MUS of size 200 out of 1445 (14%)
```

So we get an MUS of 200 clauses. Much better! (But still very big.)
It might be just about feasible to understand why the smaller set is unsatisfiable and get a proof that way (but I did not try).

Note: A different tool might find a different (smaller/bigger) MUS.

# A More Modest Goal

So for the G-S Thm, our approach unfortunately doesn't work perfectly. Let's weaken the theorem a bit to see whether that helps ...

First try: Add *unanimity* (elect $x$ if it's everyone's top alternative).

```
def cnfUnanimous():
    cnf = []
    for x in allAlternatives():
        for r in profiles(lambda r : all(top(i,x,r) for i in allVoters())):
            cnf.append([posLiteral(r,x)])
    return cnf
```

It is not overly difficult to see that surjectivity and strategyproofness actually *imply* unanimity. So adding this axiom does not really weaken the theorem. But the extra formulas might help the solver!

# Let's Try!

So now we are trying to prove the following claim:

> For $n = 2$ voters and $m = 3$ alternatives, no resolute voting rule is surjective, unanimous, strategyproof, and nondictatorial.

Let's see ...

```
>>> cnf = ( cnfAtLeastOne() + cnfResolute() + cnfSurjective()
...       + cnfUnanimous() + cnfStrategyProof() + cnfNondictatorial() )
>>> saveCNF(cnf, 'gs+una.dimacs')
>>> len(cnf)
>>> 1457
```

So that's an extra 12 clauses (*clear why?*) ...

```
$ ~/solvers/picomus gs+una.dimacs mymus.dimacs
s UNSATISFIABLE
c [picomus] computed MUS of size 96 out of 1457 (7%)
```

So we got it down from 200 to 96 clauses. Nice (but still too big).

Note: Omitting surjectivity (implied by unanimity) does not help.

# Weakening the Theorem

Geist and Peters (2017) came up with the following variant of the G-S Theorem that is logically weaker but still of some conceptual interest:

**Theorem 3 (Geist and Peters, 2017)** *No resolute voting rule for $\geqslant 3$ alternatives is majoritarian, strategyproof, and nondictatorial.*

Here a rule is called *majoritarian* if it elects an alternative whenever a strict majority ranks that alternative at the top.

Exercise: *Explain how this relates to the Condorcet Principle.*

Exercise: *Explain why this theorem is implied by the G-S Theorem.*

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

# Encoding Majoritarianism

```
def most(bools):
    return sum(bools) > len(list(bools)) / 2

def cnfMajoritarian():
    cnf = []
    for x in allAlternatives():
        for r in profiles(lambda r : most(list(top(i,x,r) for i in allVoters()))):
            cnf.append([posLiteral(r,x)])
    return cnf
```

# Let's Try!

Encode the base case of the new theorem:

```
>>> cnf = ( cnfAtLeastOne() + cnfResolute() + cnfMajoritarian()
...       + cnfStrategyProof() + cnfNondictatorial() )
>>> saveCNF(cnf, 'gp.dimacs')
```

Extract an MUS:

```
$ ~/solvers/picomus gp.dimacs mymus.dimacs
s UNSATISFIABLE
c [picomus] computed MUS of size 96 out of 1454 (7%)
```

Argh ... so this didn't help at all!

Exercise: *Explain why this attempt was bound to fail from the start.*

# Finally: Success!

But for $n = 3$ (instead of $n = 2$) and $m = 3$ it works beautifully!

So change the code:

```
n = 3
```

And then re-run everything:

```
>>> cnf = ( cnfAtLeastOne() + cnfResolute() + cnfMajoritarian()
...       + cnfStrategyProof() + cnfNondictatorial() )
>>> len(cnf)
12699
>>> saveCNF(cnf, 'gp33.dimacs')
```

```
$ ~/solvers/picomus gp33.dimacs mymus.dimacs
s UNSATISFIABLE
c [picomus] computed MUS of size 7 out of 12699 (0%)
```

# Interpreting the MUS

The file `mymus.dimacs` obtained looks like this:

```
p cnf 648 7
67 68 69 0
55 0
284 0
87 0
-68 -55 0
-67 -87 0
-69 -284 0
```

Some *ad-hoc* code to interpret literals:

```
def interpret(variable):
    r = (variable - 1) // m
    x = (variable - 1) % m
    print(str([preflist(i,r) for i in allVoters()]) + ' --> ' + str(x))
```

Let's try! Now we can make sense of the literals in the MUS:

```
>>> interpret(67)
[(2, 0, 1), (1, 2, 0), (0, 1, 2)] --> 0
```

# Understanding the Proof

The 7-clause MUS we found corresponds to this simple proof:

- in profile $\boldsymbol{R}_3 = (201, 120, 012)$ *at least one* of $0$, $1$, $2$ must win
- invoke *majoritarianism* on these three neighbouring profiles:
  - in $\boldsymbol{R}_0 = (\underline{012}, 120, 012)$ alternative $0$ must win
  - in $\boldsymbol{R}_1 = (201, 120, \underline{102})$ alternative $1$ must win
  - in $\boldsymbol{R}_2 = (201, \underline{201}, 012)$ alternative $2$ must win
- then invoke *strategyproofness* to derive these constraints:
  - $1$ wins in $\boldsymbol{R}_3 \Rightarrow 0$ loses in $\boldsymbol{R}_0$ (first voter manipulating in $\boldsymbol{R}_3$)
  - $0$ wins in $\boldsymbol{R}_3 \Rightarrow 2$ loses in $\boldsymbol{R}_2$ (second voter manipulating in $\boldsymbol{R}_3$)
  - $2$ wins in $\boldsymbol{R}_3 \Rightarrow 1$ loses in $\boldsymbol{R}_1$ (third voter manipulating in $\boldsymbol{R}_3$)
- contradiction!

Caveat: Would still have to adapt the inductive proof (seems hard!).

Note: The MUS (and thus the proof) found by Geist and Peters is slightly different (it's a bit longer).

# Exercise

You might recall that there is no resolute voting rule for $n, m = 2, 2$ that is both *anonymous* and *neutral*.

Find a proof for this fact using the SAT approach:

- encode anonymity and neutrality in CNF
- verify that the CNF is unsatisfiable and extract an MUS
- interpret the MUS to obtain a human-readable proof

Then check what happens for the following three cases:

- $n, m = 2, 3$
- $n, m = 3, 2$
- $n, m = 3, 3$

How many rules do you find (if any)? Is this what you expected?

## Summary

We now saw the full pipeline involved in using the basic SAT approach for proving impossibility theorems:

- encode axioms in CNF and generate them in the DIMACS format
- verify unsatisfiablity for the base case using a SAT solver
- extract an MUS and interpret it to obtain a human-readable proof
- prove the full theorem by induction

We also saw that it doesn't always work perfectly:

- the MUS found might be too big to interpret in practice
- the inductive proof might be quite difficult to obtain