Logics of social choice and perspectives on their software implementation

(accompanying notes for the Schloss Dagstuhl Seminar 11101)

Nicolas Troquard

University of Essex

Abstract

These are the accompanying notes to my presentation at the Schloss Dagstuhl Seminar 11101.

In this talk, I will present a logic to reason about voting procedures, proposed in a common work with W. van der Hoek and M. Wooldridge. I will discuss some perspectives on its software implementation and try to assess its practicality. I will also make a short demonstration of a prototype.

Contents

1 Introduction

Typically, the agents are the voters and the consequences will be the candidates in some election. We denote by $L(K)$ the set of *linear* orders over K. (A linear order is a relation that is transitive, antisymmetric and total.) By using a linear order, we are assuming the players cannot be indifferent between two different alternatives. A relation of *preference* is a linear order. Given K and N, a *preference profile* $<$ is a tuple $(<_i)_{i \in N}$ of preferences, where $<_i \in L(K)$ for every i. The set of preference profiles is denoted by $L(K)^N$.

Definition 1 (social choice function) *Given K and N, a* social choice function *(SCF) is a single-valued mapping from the set* $L(K)^N$ *of preference profiles into the set K of outcomes.*

For every preference profile, a social choice function describes the desirable consequence (from the point of view of the designer).

Given a set of players N and a set of consequences K, a social choice function maps a preference in $L(K)^N$ profile to a consequence in K.

2 The logic of social choice functions

In [6] (long version: [7]), we have introduced a logic for reasoning about social choice functions. The logic is evaluated over *models of social choice functions*.

Let X be an arbitrary set of propositions. We can see a *valuation* of X as a subset $V \subseteq X$ where $\mathtt{tt}$ is assigned to the propositions in V and $\mathtt{ff}$ is assigned to the propositions in $X \setminus V$. We denote the set of possible valuations over X by Θ^X.

In presence of a set of players N and a set of consequences K, the set of propositions controlled by a player $i \in N$ is defined as $At[i, K] = \{p^i_{x>y} \mid x, y \in K\}$. Every $p^i_{x>y}$ is a proposition controlled by the agent i which means that i reports that it values the consequence x at least as good as y. We also define $At[N, K] = \cup_{i \in N} At[i, K]$, which is then the set of all controlled propositions.

We can 'encode' a particular preference (or linear order) of player i as a valuation of the propositions in $At[i, K]$. However, conversely, not all valuations correspond to a linear order preference. A strategy of a player i consists of reporting a valuation of $At[i, K]$ encoding a linear order over K. For every player i, we define $strategies[i, K]$ as a set of valuations $V \in \Theta^{At[i,K]}$ such that: (1) $p^i_{x>x} \in V$, (2) if $x \neq y$ then $p^i_{x>y} \in V$ iff $p^i_{y>x} \notin V$, and (3) if $p^i_{x>y} \in V$ and $p^i_{y>z} \in V$ then $p^i_{x>z} \in V$.

For every coalition $C \subseteq N$, we note $strategies[C, K]$ the set of tuples $v_C = (v_i)_{i \in C}$ where $v_i \in strategies[i, K]$. It is the set of strategies of the coalition C. To put it another way, it corresponds to a valuation of the propositions controlled by the players in C, encoding one preference over K for every player in C.

A *state* (or reported preference profile) is an element of $strategies[N, K]$, that is, a strategy of the coalition containing all the players.

We now define the models of social choice functions.

Definition 2 (model of social choice functions) *A* model of social choice functions over N and K *(MSCF) is a tuple* $M = \langle N, K, out, (<_i)\rangle$*, such that:*

- $N = \{1, \cdots, n\}$ *is a finite nonempty set of players;*
- *K is a finite nonempty set of consequences;*

- *out is a of strategies$[N, K]$ into K;*
- *For every $i \in N$, $<_i \in L(K)$ is the true preferences of i.*

The language $\mathcal{L}^{scf}[N, K]$ is inductively defined by the following grammar:

$$\varphi \ ::= \ \top \ | \ p \ | \ x \ | \ \neg\varphi \ | \ \varphi \vee \varphi \ | \ \Diamond_C \varphi \ | \ \blacklozenge_i \varphi$$

where p is atom of $At[N, K]$, x is an atom of K, $i \in N$, and C is a coalition. $\Diamond_C \varphi$ reads that provided that the players outside C hold on to their current strategy, the coalition C has a strategy for φ. $\blacklozenge_i \varphi$ reads that i locally (at the current state) prefers a reported profile where φ is true.

Definition 3 (truth values in models of social choice functions) *Given a model of SCF $M = \langle N, K, out, (<_i)\rangle$, we are going to interpret formulas of $\mathcal{L}^{scf}[N, K]$ in a state of the model. A state $v = (v_1, \cdots, v_n)$ in M is a tuple of valuations $v_i \in strategies[i, K]$, one for each agent. The truth definition is inductively given by:*

$$\begin{array}{lll}
M, v \models p & \textit{iff} & p \in v_i \textit{ for some } i \in N \\
M, v \models x & \textit{iff} & out(v) = x \\
M, v \models \neg\varphi & \textit{iff} & M, v \not\models \varphi \\
M, v \models \varphi \vee \psi & \textit{iff} & M, v \models \varphi \textit{ or } M, v \models \psi \\
M, v \models \Diamond_C \varphi & \textit{iff} & \textit{there is a state } u \textit{ such that} \\
 & & v_i = u_i \textit{ for every } i \notin C \textit{ and } M, u \models \varphi \\
M, v \models \blacklozenge_i \varphi & \textit{iff} & \textit{there is a state } u \textit{ such that} \\
 & & out(v) <_i out(u) \textit{ and } M, u \models \varphi
\end{array}$$

The truth of φ in all models over a set of players N and a set of consequences K is denoted by $\models_{\Lambda^{scf}[N,K]} \varphi$. The classical operators $\wedge$, $\rightarrow$, $\leftrightarrow$ can be defined as usual. We also define $\Box_C \varphi \triangleq \neg\Diamond_C \neg\varphi$ and $\blacksquare_i \varphi \triangleq \neg\blacklozenge_i \neg\varphi$.

3 Game and social choice theoretical properties

In [5], we have proposed the formulation of several solution concepts in a *logic of games and propositional control* (LGPC). The logic presented in the previous section is in fact a particular logical theory of LGPC, that is, a conservative extension of LGPC. Hence, we can reuse the definitions of [5] as such.

For instance we have the following formulations:

- i's best response

  $\mathsf{BR}_i \overset{\text{def}}{=} \bigvee_{x \in K} (x \wedge \Box_i \blacklozenge_i x)$

- Nash equilibrium

  $\mathsf{NE} \overset{\text{def}}{=} \bigwedge_{i \in N} \mathsf{BR}_i$

- Dominance equilibrium

  $\mathsf{DOM} \overset{\text{def}}{=} \bigwedge_{i \in N} \Box_{N \setminus \{i\}} \mathsf{BR}_i$

We can also capture interesting properties of social choice functions, for instance:

- Citizen sovereignty

  $\mathsf{CITSOV} \overset{\text{def}}{=} \bigwedge_{x \in K} \Diamond_N x$

- Non dictatorship

  $\mathsf{NODICT} \overset{\text{def}}{=} \bigwedge_{i \in N} \Diamond_N \left(\bigvee_{x \in K} \left(x \wedge \bigvee_{y \in K \setminus \{x\}} p^i_{y > x} \right) \right)$

- Strategy-proofness

  $\mathsf{STRPROOF} \overset{\text{def}}{=} \bigwedge_{< \ \in L(K)^N} [\mathsf{true}(<) \rightarrow (\mathsf{ballot}(<) \rightarrow \mathsf{DOM})]$

$\mathsf{true}(<)$ is a reification of $<$ as the true preference profile of the players, while $\mathsf{ballot}(<)$ is a reification of $<$ as the reported preference profile of the players. We refer to [7] for details.

4 Model checking, satisfiability and synthesis

4.1 Model checking

The problem of model checking is the following.

Definition 4 (Model checking) *The problem of model checking is defined as follows.*
Instance: *A model of social choice functions M a preference profile v, and a formula φ.*
Answer: *"Yes" if $M, v \models \varphi$. "No" otherwise.*

We first show that model checking is PSPACE-complete.

Lemma 1 *Model checking is PSPACE-hard.*

Proof. *We reduce QBF into the problem of model checking. Let a quantified Boolean formula*

$$\Phi = \exists a_1 \forall a_2 \exists a_3 \cdots Q_m a_m \cdot \varphi(a_1, \cdots, a_m)$$

where Q_m is $\exists$ if m is odd and $\forall$ if m is even. Moreover, $\varphi(a_1, \cdots, a_m)$ is a propositional formula in conjunctive normal form (CNF).

We introduce a translation function $.^\sharp$ that maps a propositional formula in CNF to a formula of our logic. $.^\sharp$ is recursively defined as follows:

$$\begin{array}{lcl} a_i^\sharp & ::= & p^i_{x>y} \\ (\neg\psi)^\sharp & ::= & \neg\psi^\sharp \\ (\varphi \wedge \psi)^\sharp & ::= & \varphi^\sharp \wedge \psi^\sharp \\ (\varphi \vee \psi)^\sharp & ::= & \varphi^\sharp \vee \psi^\sharp \end{array}$$

```
1. function $eval(\langle N, K, s, (<_i)\rangle, v, \varphi)$ returns tt or ff
2.     case $\varphi \in At[N,K]$:  return $\varphi \in v$;
3.     case $\varphi \in K$:        return $\varphi = out(v)$;
4.     case $\varphi = \neg\psi$:     return not $eval(\langle N, K, s, (<_i)\rangle, v, \psi)$;
5.     case $\varphi = \psi_1 \vee \psi_2$: return $eval(\langle N, K, s, (<_i)\rangle, v, \psi_1)$ or
                                 $eval(\langle N, K, s, (<_i)\rangle, v, \psi_2)$;
6.     case $\varphi = \Diamond_C\psi$:  for every $v'_c \in strategies[C, K]$ if $eval(\langle N, K, s, (<_i)\rangle, (v_C, v'_C), \psi)$
                                 then return tt; return ff;
7.     case $\varphi = \blacklozenge_i\psi$:  for every $v' \in strategies[N, K]$ if $out(v) <_i out(v')$
                                 and $eval(\langle N, K, s, (<_i)\rangle, v', \psi)$ then return tt; return ff;
8. end
```

Figure 1: Model checking

Now, we build an MSCF $M = \langle\{1, \cdots, m\}, \{x, y\}, out, (<_i)\rangle$, where the value of out and $<$ has no importance.

$$\Phi \text{ is satisfiable iff } \Diamond_1\Box_2\Diamond_3 \cdots \{m\} \cdot [\varphi(a_1, \cdots, a_m)]^\sharp \text{ is satisfiable in } M$$

where $\{m\}$ is $\Diamond_m$ if m is odd and $\Box_m$ otherwise. Note that the preference profile v where the formula is true is not important: (i) $\Diamond_1\Box_2\psi \leftrightarrow \Diamond_N\Box_2\psi$ is a theorem for every ψ and (ii) a formula of the form $\Diamond_N\psi$ is true at one preference profile iff it is true at every preference profile.

It is readily checked that the transformation is polynomial because the outcome function out in M is arbitrary. ■

Proposition 1 *Model checking is in PSPACE.*

Proof. *For every parameter, the function eval on Figure 1 terminates because every recursive call is done with a strictly smaller formula. It is correct since it follows the semantics of the logical operators. It requires no more than $\mid \varphi \mid$ recursive calls. Moreover, in every recursion, the for loops do not require to store the results of the eval function. Hence, the algorithm runs in polynomial space.* ■

Corollary 1 *Model checking against models of social choice functions is PSPACE-complete.*

4.2 Theorem proving, satisfiability and synthesis

We are going to assume that the set of consequences and the set of voters are fixed, that is, the decision problems are indexed by K and N. Fixed sets of voters and consequences has in general a positive effect on the decidability of the logics. (See for instance [3].)

It might be quite limited for some applications. No result obtained with these parameters guarantees *a priori* that it will hold with $K + 1$ consequences or with $N + 1$

voters. However, theorem proving in small domains has recently proven tremendously promising.

In [4], Tang and Lin propose a methodology that appears to be rather general to the computer-aided proofs of theorems in social choice theory. They show how to re-prove several impossibility results. Their methodology is as follows. (i) They first prove a double induction lemma on the number of voters and consequences. (ii) They formalise the base case for 2 voters and 3 consequences in their logic. (iii) They finally use a theorem prover to check whether the base case holds.

It appears that an interesting approach to proving theorems in social choice theory is to reduce them to small base cases and use computers to conduct a search in these smaller domains. Using the same methodology, Tang and Lin are able to prove a new theorem related to Arrow's theorem

Corollary 2 *Theorem proving, satisfiability and synthesis are PSPACE-complete.*

Proof. For satisfiability we guess (undeterministically) a model of size $(| K |!)^{|N|}$ which is a (huge!) constant, independent of the input formula, and we model check it. For synthesis, the model is the output. Hence, satisfiability and synthesis are in NPSPACE = PSPACE. Then we also have that theorem proving is in CoPSPACE = PSPACE.

PSPACE-hardness of satisfiability and synthesis is obtained via the same reduction as in the proof of Lemma 1: Φ is QBF-satisfiable iff $\Diamond_1 \Box_2 \Diamond_3 \cdots \{m\} \cdot [\varphi(a_1, \cdots, a_m)]^\sharp$ is satisfiable in the logic of social choice functions. Theorem proving is then CoPSPACE-hard, hence PSPACE-hard, too. ■

5 Practical model checking

An issue concerns the size of a description of a model of social choice function.

Models of social choice functions over N and K are not compact. The description of the outcome function is in general of size exponential: There are $(| K |!)^{|N|}$ possible preference profiles. It is particularly problematic when applying the logic to model checking. The first step towards a practical solution to model checking is to represent the models in a compact way.

5.1 Positional scoring rules

In this subsection, we make use of a restricted class of models that allows us to describe a number of interesting social choice functions in a compact way: *models of positional scoring rules*. The new models are just a matter of replacing the old outcome function by a *scoring vector*.

In presence of a set of alternatives $K = \{1, \cdots, k\}$, a scoring vector is a tuple $s = (s_1, \cdots, s_k)$ where every s_i is a real, and we assume that $s_1 \geq s_2 \geq \cdots \geq s_k$ and $s_1 > s_k$. Every player is supposed to have a particular preference which is a linear order over K.

Given a preference profile $<$ over K, the consequence prescribed by the choice rule is computed as follows. Every consequence is initially given a score of 0. Then, for every player $i \in N$ and every consequence $x \in K$, if x occupies the rank r in i's preference $<_i$, then one adds s_r to the score of x. The consequence is then one among the consequences that have a maximum score: we need to pick up just one of them. It is usual to break ties by a fixed priority order on consequences [1].

The following example presents some well known voting procedures in the form of a scoring vector.

Example 1 *In presence of a set of consequences $K = \{1, \cdots, k\}$:*

$$\begin{array}{lcl} \textit{Plurality} & : & (1, 0, \cdots, 0) \\ \textit{Veto} & : & (1, \cdots, 1, 0) \\ \textit{Borda} & : & (k-1, k-2, \cdots, 1, 0) \end{array}$$

We now define the models of positional scoring rules.

Definition 5 (model of positional scoring rule) *A* model of positional scoring rule over N and K *(MPSR) is a tuple $M = \langle N, K, s, (<_i)\rangle$, such that:*

- *$N = \{1, \cdots, n\}$ is a finite nonempty set of players;*
- *K is a finite nonempty set of consequences;*
- *s is a scoring vector over K;*
- *For every $i \in N$, $<_i \in L(K)$ is the true preferences of i.*

We note $s(v)$ the consequence in K selected by the score vector s for a preference profile v. Computing the scores of every consequences can be done in time $O(k \cdot n)$. Breaking ties is usually assumed to be computationally easy and we will assume so in this note. It is then easy to compute $s(v)$ for every v.

Lemma 2 *For every preference profile v the complexity of computing $s(v)$ is $O(k \cdot n)$*

Definition 6 (truth values in models of positional scoring rules) *Given an MPSR $M = \langle N, K, s, (<_i)\rangle$, we are going to interpret formulas of $\mathcal{L}^{scf}[N, K]$ in a state of the model. A state $v = (v_1, \cdots, v_n)$ in M is a tuple of valuations $v_i \in strategies[i, K]$, one for each agent. The truth definition is similar to definition 3 except for:*

$$\begin{array}{lll} M, v \models x & \textit{iff} & s(v) = x \\ M, v \models \blacklozenge_i \varphi & \textit{iff} & \textit{there is a state } u \textit{ such that} \\ & & s(v) <_i s(u) \textit{ and } M, u \models \varphi \end{array}$$

Corollary 3 *The problem of model checking against a model of positional scoring rules is PSPACE-complete.*

Proof. *Model checking against positional scoring rules can be done by a straightforward adaptation of the algorithm of Figure 1, replacing the out functions by functions s. Because computing s is easy (Lemma 2), the problem is in PSPACE.*

Hardness follows from Lemma 1. ■

5.2 Generalised scoring rules

In [8], Xia and Conitzer introduce a class of social choice functions that they call *generalised scoring rules*. They are compact representations of a large class of social choice functions including positional scoring rules, STV, Copeland, etc.[1]

We present the generalised scoring rules, but do not enter into much details. We do not include them either in the software implementation presented in the remaining sections.

Let $c \in \mathbb{N}$ and let $P = \{P_1, \cdots, P_q\}$ be a partition of $C = \{1, \cdots, c\}$.

For every $a, b \in \mathbb{R}^c$, we say a and b are *equivalent with respect to P*, denoted $a \approx_P b$, if for any $l \leq q$, any $i, j \in P_l$, we have $a_i \geq a_j$ iff $b_i \geq b_j$.

We say that a function g is *compatible* with P if for any $a, b \in \mathbb{R}^c$, if $a \approx_P b$ then $g(a) = g(b)$.

Definition 7 (generalised socring rules) *Let $f : L(K) \longrightarrow \mathbb{R}^c$ and $g : \mathbb{R}^c \longrightarrow K$. The functions f and g determine the generalised scoring rule $GS(f, g)$:*

For every preference profile $(v_1, \cdots, v_n) \in L(K)^N$,

$$GS(f, g)((v_1, \cdots, v_n)) = g(\Sigma_1^n f(v_i))$$

We now define the models of generalised scoring rules.

Definition 8 (model of generalised scoring rule) *A* model of generalised scoring rule over *N* and *K (MGSR) is a tuple $M = \langle N, K, (c, P, f, g), (<_i)\rangle$, such that:*

- *$N = \{1, \cdots, n\}$ is a finite nonempty set of players;*
- *K is a finite nonempty set of consequences;*
- *$c \in \mathbb{N}$;*
- *P is a partition of $\{1, \cdots, c\}$;*
- *$f : L(K) \longrightarrow \mathbb{R}^c$;*
- *$g : \mathbb{R}^c \longrightarrow K$;*
- *For every $i \in N$, $<_i \in L(K)$ is the true preferences of i.*

Definition 9 (truth values in models of generalised scoring rules) *Given an MGSR $M = \langle N, K, (c, P, f, g), (<_i)\rangle$, we are going to interpret formulas of $\mathcal{L}^{scf}[N, K]$ in a state of the model. A state $v = (v_1, \cdots, v_n)$ in M is a tuple of valuations $v_i \in strategies[i, K]$, one for each agent. The truth definition is similar to definition 3 except for:*

$$\begin{array}{lll} M, v \models x & \textit{iff} & GS(f, g)(v) = x \\ M, v \models \blacklozenge_i \varphi & \textit{iff} & \text{there is a state } u \text{ such that} \\ & & GS(f, g)(v) <_i GS(f, g)(u) \text{ and } M, u \models \varphi \end{array}$$

We briefly comment on the complexity of model checking models of generalised scoring rules

[1]Xia and Conitzer mention in the article that they are not aware of a commonly studied social choice function that cannot be represented as a generalised scoring rule.

The complexity of model checking the logic against models of generalised scoring rules will depend on the complexity of computing $GS(f, g)$. It can happen that it is easy to compute. On the other side of the range, it can also happen that it is undecidable. Given a function *fun*, we note $M(fun)$ its complexity class.

Assuming that we have an oracle for solving GS in unit time, one can solve model checking against a model of generalised scoring rule in polynomial space. That is, model checking against a model of generalised scoring rule with function GS is in $\mathrm{PSPACE}^{M(GS)}$.

The problem is PSPACE-hard whatever the computational complexity of GS. Hence, whenever $M(GS) \subseteq \mathrm{PSPACE}$, we have that model checking against a generalised scoring rule with function GS is PSPACE-complete (because $\mathrm{PSPACE}^{PSPACE} = \mathrm{PSPACE}$).

In order to obtain a more interesting result, we might want to represent the mathematical functions f and g as *computational trees* [2].

6 Implementation

We have made a Common Lisp implementation of a prototype for the logic of social choice functions, using the models of positional scoring rules.

We adopted some convenient conventions. The set of players is represented by an integer p and players then range from 0 to $p-1$. The set of consequences is represented by an integer c and consequences then range from 0 to $c-1$.

We assume that a formula is accepted by the following BNF:

```
F  ::=      (x)        |    (i (x x))    |     (neg F)      |
        (and F F ...)  |  (or F F ...)   |  (diamond i F)   |
          (box i F)    |  (pospref i F)  |  (necpref i F)
```

where `x` is a consequence, `i` is a player.

Example 2 *A formula characterising the states where player* 1 *reported preferring consequence* 0 *over consequence* 1*, and where player* 0 *can change unilaterally its strategy in a way to ensure the consequence* 1 *is given as follows:*

```
(and (1 (0 1))
     (diamond 0 (1)))
```

Example 3 *A Nash equilibrium with two players and two consequences is represented by the following formula:*

```
(and
 (or
  (and (1) (box 1 (pospref 1 (1))))
  (and (0) (box 1 (pospref 1 (0)))))
 (or
  (and (1) (box 0 (pospref 0 (1))))
  (and (0) (box 0 (pospref 0 (0))))))
```

It is worth noting that formulas representing equilibria are of primary use, but can be tedious to write at hand. For this reason, we have written a function `(init-gt p c)` that generates the formulas `ne` and `dom` representing respectively a Nash equilibrium and a dominance equilibrium for a number p of players and c consequences.

7 Example of execution

Load the program tools in the environment.[2]

```
[1]> (load 'scf.lisp)
;; Loading file scf.lisp ...
;; Loaded file scf.lisp
T
```

Define the social choice function `scf32` over the set of players $\{0, 1, 2\}$ and the set of consequences $\{0, 1\}$. It is defined as the positional scoring rule $(1, 0)$. In other words, it is the plurality rule with two consequences.

```
[2]> (defvar  scf32 (make-scf :players 3 :consequences 2 :scorerule '(1 0)))
#S(SCF :PLAYERS 3 :CONSEQUENCES 2 :SCORERULE (1 0))
```

Define `profile32` as the preference profile where player 0 prefers consequence 0 over consequence 1; player 1 prefers 1 over 0; player 2 prefers 0 over 1.

```
[3]> (defvar profile32 '((0 1) (1 0) (0 1)))
((0 1) (1 0) (0 1))
```

Define the model of positional scoring rule `mscf32` as the social choice function `scf32` with `profile32` as the true preference profile.

```
[4]> (defvar mscf32 (make-model-scf :scfunc scf32 :prefprofile profile32))
#S(MODEL-SCF :SCFUNC #S(SCF :PLAYERS 3 :CONSEQUENCES 2 :SCORERULE (1 0))
   :PREFPROFILE ((0 1) (1 0) (0 1)))
```

Initialise the solution concepts with 3 players and 2 consequences.

```
[5]> (init-gt 3 2)
T
```

Find the dominance equilibria on `mscf32`. The profile `profile32` is the only solution.

```
[6]> (solve dom mscf32)
(((0 1) (1 0) (0 1)))
```

Check that `scf32` is strategy-proof.

```
[7]> (is-strategy-proof scf32)
T
```

Define the social choice function `scf33` over the set of players $\{0, 1, 2\}$ and the set of consequences $\{0, 1, 2\}$. It is defined as the positional scoring rule $(2, 1, 0)$. In other words, it is the Borda rule with three consequences.

```
[8]> (defvar  scf33 (make-scf :players 3 :consequences 3 :scorerule '(2 1 0)))
#S(SCF :PLAYERS 3 :CONSEQUENCES 3 :SCORERULE (2 1 0))
```

Define `profile33` as the preference profile where: $1 <_0 0 <_0 2$, $0 <_1 1 <_1 2$, and $2 <_2 1 <_2 0$.

```
[9]> (defvar profile33 '((2 0 1) (2 1 0) (0 1 2)))
((2 0 1) (2 1 0) (0 1 2))
```

Define the model of positional scoring rule `mscf33` as the social choice function `scf33` with `profile33` as the true preference profile.

```
[10]> (defvar mscf33 (make-model-scf :scfunc scf33 :prefprofile profile33))
#S(MODEL-SCF :SCFUNC #S(SCF :PLAYERS 3 :CONSEQUENCES 3 :SCORERULE (2 1 0))
   :PREFPROFILE ((2 0 1) (2 1 0) (0 1 2)))
```

[2]This first instruction is for a clisp implementation of Common Lisp. It might differ for other implementations. However, the rest of the instructions should be completely compatible.

Initialise the solution concepts with 3 players and 3 consequences.

```
[11]> (init-gt 3 3)
T
```

Model check the dominance equilibria on `mscf33` at the profile `profile33`. It is *not* a dominance equilibrium.

```
[12]> (model-check dom mscf33 profile33)
NIL
```

Model check the Nash equilibria on `mscf33` at the profile `profile33`. It is a Nash equilibrium.

```
[13]> (model-check ne mscf33 profile33)
T
```

Find the Nash equilibria on `mscf33`. We use the optional parameter `t` as the *verbose mode* that prints the solutions on the fly.

```
[14]> (solve ne mscf33 t)

((2 1 0) (2 0 1) (2 1 0))
((2 1 0) (2 0 1) (2 0 1))
((2 1 0) (2 0 1) (1 2 0))
((2 1 0) (2 0 1) (1 0 2))
((2 1 0) (2 0 1) (0 2 1))
((2 1 0) (2 0 1) (0 1 2))
((2 0 1) (2 1 0) (2 1 0))
((2 0 1) (2 1 0) (2 0 1))
((2 0 1) (2 1 0) (1 2 0))
((2 0 1) (2 1 0) (1 0 2))
((2 0 1) (2 1 0) (0 2 1))
((2 0 1) (2 1 0) (0 1 2))
((1 2 0) (1 2 0) (1 2 0))
((1 2 0) (1 2 0) (1 0 2))
((1 2 0) (1 0 2) (1 2 0))
((1 0 2) (1 2 0) (1 2 0))
((1 0 2) (1 2 0) (1 0 2))
((0 2 1) (1 2 0) (0 1 2))
((0 2 1) (1 0 2) (0 2 1))
((0 2 1) (1 0 2) (0 1 2))
((0 2 1) (0 2 1) (0 2 1))
((0 2 1) (0 2 1) (0 1 2))
((0 2 1) (0 1 2) (0 2 1))
((0 2 1) (0 1 2) (0 1 2))
((0 1 2) (1 2 0) (0 1 2))
((0 1 2) (1 0 2) (0 2 1))
((0 1 2) (1 0 2) (0 1 2))
((0 1 2) (0 2 1) (0 2 1))
((0 1 2) (0 2 1) (0 1 2))
((0 1 2) (0 1 2) (0 2 1))
((0 1 2) (0 1 2) (0 1 2))
(((0 1 2) (0 1 2) (0 1 2)) ((0 1 2) (0 1 2) (0 2 1)) ((0 1 2) (0 2 1) (0 1 2))
 ((0 1 2) (0 2 1) (0 2 1)) ((0 1 2) (1 0 2) (0 1 2)) ((0 1 2) (1 0 2) (0 2 1))
 ((0 1 2) (1 2 0) (0 1 2)) ((0 2 1) (0 1 2) (0 1 2)) ((0 2 1) (0 1 2) (0 2 1))
 ((0 2 1) (0 2 1) (0 1 2)) ((0 2 1) (0 2 1) (0 2 1)) ((0 2 1) (1 0 2) (0 1 2))
 ((0 2 1) (1 0 2) (0 2 1)) ((0 2 1) (1 2 0) (0 1 2)) ((1 0 2) (1 2 0) (1 0 2))
 ((1 0 2) (1 2 0) (1 2 0)) ((1 2 0) (1 0 2) (1 2 0)) ((1 2 0) (1 2 0) (1 0 2))
 ((1 2 0) (1 2 0) (1 2 0)) ((2 0 1) (2 1 0) (0 1 2)) ((2 0 1) (2 1 0) (0 2 1))
 ((2 0 1) (2 1 0) (1 0 2)) ((2 0 1) (2 1 0) (1 2 0)) ((2 0 1) (2 1 0) (2 0 1))
 ((2 0 1) (2 1 0) (2 1 0)) ((2 1 0) (2 0 1) (0 1 2)) ((2 1 0) (2 0 1) (0 2 1))
 ((2 1 0) (2 0 1) (1 0 2)) ((2 1 0) (2 0 1) (1 2 0)) ((2 1 0) (2 0 1) (2 0 1))
 ((2 1 0) (2 0 1) (2 1 0)))
```

Find in `mscf33` the Nash equilibria where player 1 prefers the consequence 2 over the consequence 1.[3] We don't use the verbose mode.

[3]Some remarks specific to LISP are due. Note the use of a comma in the term `,ne` to force the evaluation of the formula representing a Nash equilibrium as initialised by `(init-gt ...)`, that is, to expand the

```
[15]> (solve ‘(and ,ne (1 (2 1))) mscf33)
(((0 1 2) (0 2 1) (0 1 2)) ((0 1 2) (0 2 1) (0 2 1)) ((0 2 1) (0 2 1) (0 1 2))
 ((0 2 1) (0 2 1) (0 2 1)) ((2 0 1) (2 1 0) (0 1 2)) ((2 0 1) (2 1 0) (0 2 1))
 ((2 0 1) (2 1 0) (1 0 2)) ((2 0 1) (2 1 0) (1 2 0)) ((2 0 1) (2 1 0) (2 0 1))
 ((2 0 1) (2 1 0) (2 1 0)) ((2 1 0) (2 0 1) (0 1 2)) ((2 1 0) (2 0 1) (0 2 1))
 ((2 1 0) (2 0 1) (1 0 2)) ((2 1 0) (2 0 1) (1 2 0)) ((2 1 0) (2 0 1) (2 0 1))
 ((2 1 0) (2 0 1) (2 1 0)))
```

We check that `scf33` is not strategy-proof.

```
[16]> (is-strategy-proof scf33)
NIL
```

References

[1] S. Brams and P. Fishburn. *Handbook of Social Choice and Welfare*, volume 1, chapter Voting Procedures, pages 175–236. Elsevier, 2002.

[2] Peter Bürgisser, Michael Clausen, and Mohammad A. Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

[3] Marc Pauly. Social Choice in First-Order Logic: Investigating Decidability & Definability. Unpublished.

[4] Pingzhong Tang and Fangzhen Lin. Computer-aided proofs of Arrow's and other impossibility theorems. *Artificial Intelligence*, 173(11):1041–1053, 2009.

[5] N. Troquard, W. van der Hoek, and M. Wooldridge. A logic of games and propositional control. In *AAMAS'09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 961–968. IFAAMAS, 2009.

[6] N. Troquard, W. van der Hoek, and M. Wooldridge. A logic of propositional control for truthful implementations. In *TARK'09: Proceedings of the 12th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 237–246, New York, NY, USA, 2009. ACM.

[7] N. Troquard, W. van der Hoek, and M. Wooldridge. Reasoning about Social Choice Functions. *Journal of Philosophical Logic*, 2011. To appear. Also published as arXiv:1102.3341v1.

[8] L. Xia and V. Conitzer. Generalized scoring rules and the frequency of coalitional manipulability. In *EC '08: Proceedings of the 9th ACM conference on Electronic commerce*, pages 109–118, New York, NY, USA, 2008. ACM.

macro `ne`. For this reason, we use a back-quote ‘ instead of a quote ’ to suppress the evaluation of the list representing the formula we want to solve.