Covertly Controlling Choices

Manipulating Decision Making Under Partial Knowledge

MSc Thesis *(Afstudeerscriptie)*

written by

Simone Griffioen
(born November 18, 1992 in Groningen)

under the supervision of Prof. dr. Jan van Eijck, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
June 15, 2017	Prof. dr. Ronald de Wolf (Chair)
	Prof. dr. Jan van Eijck (Supervisor)
	Dr. Ulle Endriss
	Dr. Ronald de Haan
	Malvin Gattinger, MSc

Institute for Logic, Language and Computation

Abstract

The problem of reaching collective decisions on interconnected issues is studied in binary judgement aggregation with integrity constraints. In this thesis, we present an implementation of the framework for binary judgement aggregation and extend this implementation to study the decision making process. First we introduce several (new) rules, such as the priority rule, the distance based rule and the least squares rule. We illustrate and motivate the usefulness of these rules by examples using the implementation. We introduce the notion of a majority preserving rule and we prove that both the priority and the distance based rule are majority preserving. It is well known that most reasonable aggregation rules are susceptible to manipulation under full knowledge. We study manipulation of the decision making process. To generalize this, we quantify the preferences of agents and we use this to study manipulations under both full, but also partial knowledge. We illustrate how agents could manipulate certain aggregation rules. Not only aggregation rules can be manipulated, other parts of the decision making process are susceptible to manipulation as well. An agent could use her knowledge for making a choice of an aggregation rule or when setting an agenda. We adapt a known concept of agenda manipulation to our setting and combine this with manipulation of the aggregation rule. We investigate how much knowledge agents need to be able to manipulate and we show that even without knowledge, there are plenty of opportunities for manipulation. Finally, we use the quantified preferences to quantify manipulability of aggregation rules under different amounts of knowledge, and we show how we can use this quantification to choose a rule that is less manipulable.

Acknowledgements

I would like to express my gratitude to everyone who has made this thesis possible. The positive feedback of my friends, fellow students and family, who were willing to listen to and discuss my ideas encouraged me to stay enthusiastic and it helped me to get a clear line in my research. Malvin, Noor and my dad read and annotated my thesis. They pointed out what I should clarify and where a nice example would help a lot. Sharing thesis suffering with Hugo was motivating, since suffering together is much nicer than suffering alone. Jan was always willing to have a chat about my thesis, politics or any other interesting subject. He organized weekly thesis lunches in which I could share my thesis troubles with other students in the same position. Furthermore I would like to acknowledge the members of the committee for the time they took to read my thesis and prepare the defense.

Contents

1 Introduction

1.1 Motivation

Each day every one of us has to make many decisions.

"What do I want for dinner tonight?"
"What color shoes do I want to buy?"
"Which movie do I want to see?"
"Do I want to go to bed early tonight?"

These decisions might be interconnected. For example, if I want to go to bed early tonight, I do not want to see a very long movie. Making all these decisions on your own can already be a tough task, however, if multiple people are involved, reaching a collective decision is certainly not a piece of cake! We will illustrate this with an example.

Example 1 - Cat or Dog. A family has to decide on the purchase of a new pet. In Table 1.1, the family members' preferences about the purchase can be found.

	dog	cat	both
Member 0	Yes	No	No
Member 1	Yes	Yes	Yes
Member 2	No	Yes	No
Majority	Yes	Yes	No

Table 1.1: The opinion of 3 family members on the purchase of pets.[1]

When making binary decisions, a reasonable strategy is to follow the opinion of the majority. This way, if the family would vote for having both a cat and a dog, they would decide not to. However, if they would vote for having a cat, they would buy a cat, and if they would continue to vote for having a dog, they would also buy a dog. In both cases, the decision is made according to the majority, however, we end up with contradictory outcomes: according to majority, they should buy a cat, and a dog, but not both a cat and a dog. This is impossible! ■

Note that while all the opinions of the individual family members are completely rational, proposition-wise majority voting gives us a result that is not. That is, the outcome is logically impossible. This is

[1] It might seem quite unnatural to vote for these three issues, however this example shows that it actually matters whether we vote separately on the first two issues, or only on the third.

also known as the doctrinal paradox. Clearly reaching a decision on these kinds of subjects is not as straightforward as it might look at first sight. How to do this *judgement aggregation* is a question that generalizes earlier preference aggregation problems, in which a winner from a set of alternatives has to be chosen.[2] A paradox in judgement aggregation is a case in which all individuals stick to a certain constraint, but a seemingly reasonable aggregation rule produces a group decision that violates this same constraint, such as in the example above. The field of judgement aggregation is concerned with these paradoxes and finding aggregation rules that do not generate paradoxes, that is, rules that *lift* the constraint to the collective level. This is mostly studied by the axiomatic method. Here axioms are given that describe desirable properties of aggregation rules. Grandi and Endriss (2011a) showed that some very desired axioms can only hold simultaneously when the rule is a quota rule. In their paper, the class of integrity constraints that are lifted by quota rules is characterised. Most integrity constraints are not lifted by quota rules. For such an integrity constraint, all reasonable rules are susceptible to manipulation, that is, people may want to lie about their preferences to obtain better outcomes of the aggregation rule. This is manipulation of the aggregation rule. In the literature, there are several definitions for manipulability of an aggregation rule, which are not all equivalent. For example, Dietrich and List (2007b) uses another definition than Grandi (2012). Besides manipulation of the aggregation rule, there are other parts in the decision making process where an agent could influence the outcome. For one, the individuals involved in the decision making often first have to agree on the rule that they use. In daily life, mostly little consideration is given to this: a certain rule is proposed and accepted if it sounds reasonable. However, as we will see, picking a certain rule can be a way of manipulating the outcome. Besides this, if a certain type of rule is chosen, the outcome will also depend on the agenda setting, that is: the questions that are voted on and the order of voting on them. One can sometimes set the agenda in such a way that a more preferable outcome will be obtained. This is called agenda manipulation. Dietrich (2013), studies agenda manipulation for binary aggregation.

These different kinds of manipulation that can occur within the decision making process have been studied before, but in most research not much regard is given to the case where the manipulator has only partial knowledge. In every day life, agents will almost never have full knowledge about the preferences of others. Therefore, in this thesis, we will study how the decision making process can be manipulated under partial knowledge. We will discuss several notions of manipulability and we will quantify the manipulability of some rules.

We study manipulations for agents that are not risk-averse. They only care about their expected happiness. A different approach is taken in Terzopoulou (2017). Here it is assumed that agents are risk averse and only manipulate if they are sure they will be better off.

One might wonder, is manipulability actually important? Of course in some cases it might not be. If you know that you can trust your co-voters to tell the truth it will not matter. However, in other cases, manipulability is mostly something you want to avoid. Even though it seems that everyone has the same advantage if a rule is manipulable, this is not really the case. First of all, not everyone is equally good with numbers, and since manipulation asks for some calculations - even more so in case of manipulation under partial knowledge - for some people it is easier to manipulate than for others. Secondly, if a rule is manipulable, it does not mean that everyone is better off when they lie. People with certain preferences may have much more opportunities to manipulate than others. Another reason for wanting to avoid manipulability is that we would like to know the individuals' true preferences. Otherwise it might be that everyone is trying to manipulate, producing an outcome that has almost nothing to do with everyones true preferences.

[2] In Grandi and Endriss (2011a), a framework for *binary judgement aggregation with integrity constraints* is presented and a translation from the framework of preference aggregation into that of binary judgement aggregation is given.

1.2 Outline of thesis

The remainder of this thesis is structured as follows.

In Chapter 2 we study the basics of binary aggregation with integrity constraints. The first section of the first chapter introduces the necessary definitions for binary aggregation from Grandi and Endriss (2011a). We give a functional implementation of this framework that is based on Eijck (2016)[3]. We illustrate the implementation by means of multiple examples. In the second section we introduce several aggregation rules together with their implementation and we discuss their properties using the axiomatic method. In this, we adapt the standard axioms to work for non-resolute rules. We prove that some rules have the nice property of agreeing with the majority rule when possible. Some examples are given to illustrate relevance of different rules. In the third section we briefly explain how we can translate preference aggregation into the framework of binary judgement aggregation. We illustrate this by an example and we describe the Copeland rule as an example of a social choice function.

The topic of Chapter 3 is manipulating decision making. The first and largest section of this chapter is devoted to manipulation of the aggregation rule. When agents are provided with only partial knowledge about the preferences of the others, they try to maximize their expected "satisfaction with the outcome" by lying about their preferences. This concept is formalized using game theoretic utility functions. We use these to give different definitions of manipulation of the parts of the decision making process (under partial knowledge). We provide a manipulation tool an agent can use to determine her optimal vote. We prove some characterization results on manipulability of the aggregation rule.

In the second section of Chapter 3 we discuss how agents can manipulate the setting of the aggregation procedure. There are multiple ways in which they can do this. They can influence the choice of an aggregation rule and they can influence the agenda setting. For this we translate the framework of Agenda manipulation from Dietrich (2013) to the framework from Grandi and Endriss (2011a). We introduce agenda manipulation under partial knowledge. We combine agenda manipulation with manipulation of the aggregation rule and we implement a function that determines the optimal agenda for a given agent with a given utility.

In Chapter 4 we look at manipulability from another perspective. We investigate which rules are more or less susceptible to manipulation. A standard way to quantify manipulability of aggregation rules is the percentage of profiles in which an agent with full knowledge could manipulate. We propose a different measure that takes into account how beneficial a certain manipulation is for an agent. If we apply this on a large number of random utility functions, we can determine an average percentual win in utility for manipulators. We apply this on a specific example rule for a certain integrity constraint.

Chapter 5 concludes and gives several suggestions for further research.

A bonus feature of this thesis is that all aggregation rules discussed or newly proposed are implemented. Indeed, the implementation is very useful in guiding our analysis. Two appendices give details on basic functions that the implementation depends on. The thesis should be accessible without consultation of the appendices. These are merely provided to ensure a self-contained presentation of the implementation.

[3]All implementations used and given in this thesis can be found on Github (`https://github.com/simonegrif/CovertlyControllingChoices/`).

List of Symbols

$(F_A)_{A\in\mathcal{A}}$	An aggregation system, containing an aggregation rule for each feasible agenda.
$>^{\mu}$	Pairwise majority comparison.
$\mathsf{Agree}(B, B')$	The set of issues $j \in \mathcal{I}$ for which $b_j = b'_j$.
χ	The set of alternatives in preference aggregation.
DB	The distance based rule.
IC	An integrity constraint.
$\mathrm{IC}_<$	The integrity constraint for preference aggregation.
$\mathsf{Maj}(\mathbf{B})$	The rule that gives those ballots B that agree on each issue either with the weak or the strict majority.
$\mathsf{Maj}_S(\mathbf{B})$	The strict majority resolution of profile $\mathbf{B}$
$\mathsf{Maj}_W(\mathbf{B})$	The weak majority resolution of profile $\mathbf{B}$
$\mathcal{A}$	The set of feasible agendas.
$\mathcal{E}(S)$	The set of ballots over Ω that are consistent with some ballot in S.
$\mathcal{I}$	A set of issues.
$\mathcal{N}$	A set of individual agents.
$\mathsf{Mod}(\mathrm{IC})$	The list of ballots satisfying IC.
Ω	A set of issues of which an agenda should be a subset.
PR	The priority rule.
$\mathbf{B}$	A profile, consisting of a ballot B_i for each agent i.
$\mathbf{B}_{-i}$	profile $\mathbf{B}$ without ballot B_i
B_i	The ballot of agent i from profile $\mathbf{B}$.
b_j	The vote for issue j in ballot B.
$B_{i,j}$	The vote of agent i on issue j in profile $\mathbf{B}$.
$E_F(U_i, B, k)$	The expected utility U_i of the outcome of F when casting ballot B, given knowledge k.

l_0	The integrity constraint $p_0 \wedge p_1 \rightarrow (\neg p_2 \wedge \ldots \wedge \neg p_{m-1})$
LS	The least squares rule.
MM	The minimax rule.
n_0	The empty integrity constraint.
p_j	The propositional symbol associated with issue j
q_0	The integrity constraint $(p_0 \wedge p_2) \leftrightarrow p_1$

2 | Binary aggregation with integrity constraints

This chapter will provide basic definitions, illustrated by an implementation. We use Haskell as implementation language. Some familiarity with functional programming notation is assumed in our presentation. For the required background in functional programming the reader can consult Khan (2014). The thesis is written in so-called Literate Programming style. See Knuth (1992).

2.1 Basic definitions

We consider a set of individual agents $\mathcal{N} = \{0, \ldots, n-1\}$. All agents give their opinion on a set of (binary) issues $\mathcal{I} = \{0, \ldots, m-1\}$ by means of casting a boolean vote.

```
module Aggregation where
import MyBasics
import Data.List

type Vote = Bool
type Issue = Int
type Agent = Int
```

An agent can express his opinions on this set of issues by casting a Ballot, which is an element of the boolean combinatorial domain $\{\mathsf{True}, \mathsf{False}\}^m$.

```
type Ballot = [Vote]
```

Issues are the indices of ballots. Since a ballot is quite spacious, we will sometimes convert this to an integer by converting the binary representation into an integer.[1]

[1] To improve readability, some of the functions can be found in Appendix A.

```
type BalInt = Int

b2i :: Ballot -> BalInt
b2i = bin2int
```

To convert an integer back to a ballot, we can convert the integer to a list of bits. We map 1 to True and 0 to False. We need to know the number of issues such that we can add the leading zeros (for `[0,0,1,1]` is different from `[1,1]`). If the integer is too large to generate a ballot of given length we generate an error.

```
i2b :: Int -> BalInt -> Ballot
i2b m = assert
  (\ i _ -> i < 2^m)
  (\ i _ -> "ballot "++ show i ++" does not exist")
  (\ i -> let
    bs = int2bin i
    m' = m - length bs
  in
    replicate m' False ++ bs)
```

We can convert the ballot (True, True, False) to an integer and back to a ballot.

```
  > b2i [True,True,False]
6
  > i2b 3 6
[True,True,False]
```

A profile $\mathbf{B} = (B_1, \ldots B_n)$ consists of a ballot for each agent.

```
type Profile = [Ballot]
type ProfInt = [BalInt]
```

We can convert a list of ballots to a list of integer ballots and vice versa.

```
is2bs :: Int -> [BalInt] -> [Ballot]
is2bs m = map (i2b m)

bs2is :: [Ballot] -> [BalInt]
bs2is = map b2i
```

Note that it is assumed for this conversion that the ballots are all of the same size. We write b_j for the $(j-1)^{\text{th}}$ element of a ballot B (so the vote on issue j in ballot B) and $b_{i,j}$ for the $(j-1)^{\text{th}}$ element of ballot B_i within the profile. Agents are the indices of the profile. When S is a set of ballots, we define

$S_j = \{v \in \{\mathsf{True}, \mathsf{False}\} \mid \exists B \in S \text{ such that } b_j = v\}$. We will write $\mathbf{B}_{-i}$ for the partial profile without ballot B_i, which we will call the $\min_i$-profile. We write $(B, \mathbf{B}_{-i})$ for the profile in which the ballot of agent i is replaced by ballot B.

```
type MinIProfile = (Profile, Profile)
```

We can split a profile to obtain the corresponding $\min_i$-profile.

```
splitProfile :: Agent -> Profile -> MinIProfile
splitProfile i profile = (take i profile, drop (i+1) profile)
```

We can use our implementation to represent any profile, such as the profile in Example 1.

Example 1 (continued). We continue with the example from the introduction. In Table 2.1, we repeat the family members preferences about the purchase. In our implementation this profile is represented as `[4,7,1]`.

Issue:	0 : dog	1 : dog and cat	2 : cat
Member 0	True	False	False
Member 1	True	True	True
Member 2	False	False	True

Table 2.1: Three family members cast their vote on three issues.[2]

```
exampleA :: Profile
exampleA = is2bs 3 [4,7,1]
```

```
  > exampleA
[[True,False,False],[True,True,True],[False,False,True]]
```

■

If we know the number of issues, we can generate the list of all possible ballots on those issues.

```
allBallots :: Int -> [Ballot]
allBallots 0 = [[]]
allBallots m = map (True :) allbs ++
               map (False :) allbs where
                 allbs = allBallots (m-1)
```

[2]Note that the ordering of the issues in this example seems somewhat odd. This is done with eye on a future use in which the ordering will be important.

Two ballots B and B' are said to *agree* on an issue i if $b_i = b'_i$. Two agents agree on an issue if the ballots representing their true preferences agree on that issue.

```
agree :: Ballot -> Ballot -> Issue -> Bool
agree b1 b2 i = b1 !! i ==  b2 !! i
```

We define the set

$$\mathsf{Agree}(B, B') = \{j \in \mathcal{I} \mid b_j = b'_j\},$$

of issues on which two ballots agree.

```
agreeSet :: Ballot -> Ballot -> [Issue]
agreeSet b1 b2 = let
    m = length b1
  in [i | i <- [0..(m-1)], b1 !! i == b2 !! i]
```

The coalition of an issue, given a profile, is the list of agents that are in favour of that issue (i.e., the agents that have voted True for the issue).

```
coalition :: Issue -> Profile -> [Agent]
coalition i profile = let
    pairs = zip [0..] profile
  in
    [ ag | (ag,ballot) <- pairs, ballot!!i ]
```

The anticoalition is the complement of the coalition.

```
anticoalition ::  Issue -> Profile -> [Agent]
anticoalition i profile = let
    n = length profile
  in
    [0..n-1] \\ coalition i profile
```

Integrity constraints

In Example 1, any rational decision cannot include a False for "cat and dog" while including True for both "cat" and "dog". This poses a constraint on the outcome of the aggregation procedure, and if we expect people to be rational, it also poses a constraint on cast ballots. In other examples there might be different constraints expressing the coherence of certain issues. To express these constraints, we associate a propositional symbol p_j with each issue j, and let $\mathcal{L}_{PS}$ be the corresponding propositional language.

Definition 1. An *Integrity constraint* is a formula $\mathrm{IC} \in \mathcal{L}_{PS}$ that expresses the connection between issues.

A ballot B is said to satisfy an integrity constraint IC when the formula we obtain by substituting every p_i in IC by b_i is not equivalent to falsum. We will use integrity constraints only to check whether certain ballots satisfy them, hence we will implement them as a property of ballots instead of as a logical formula.

```
type IC = Ballot -> Bool
```

For example, we can have the following integrity constraints:

- $n_0 = \top$ is the empty constraint,
- $s_0 = p_0 \leftrightarrow p_1$ says that the first two issues need to have the same value,
- $q_0 = (p_0 \wedge p_2) \leftrightarrow p_1$ states that issues 0 and 2 are both true if and only if issue 1 is,
- $r_0 = \neg(p_0 \wedge p_1 \wedge p_2)$ expresses that the first three issues cannot all be true,
- $v_0 = p_0 \vee \ldots \vee p_{m-1}$ expresses that not all issues should be false,
- $l_0 = p_0 \wedge p_1 \rightarrow (\neg p_2 \wedge \ldots \wedge \neg p_{m-1})$ expresses that if the first two issues are true, then all others should be false,
- $z_0 = p_0 \vee p_1$ expresses that one of the first two issues should be true,

which are encoded below.

```
n0, s0, q0, r0, v0, l0, z0 :: IC
n0 _          = True
s0 (x:y:_)    = x == y
s0 _          = True -- True for single issue ballots
q0 (x:y:z:_) = ((x && z) --> y) && (y --> (x && z))
q0 _          = True -- True for 1 and 2 issue ballots
r0 (x:y:z:_) = not (x && y && z)
r0 _          = True -- True for 1 and 2 issue ballots
v0            = or
l0 (x:y:xs)     = (x && y) --> all not xs
l0 _          = True -- True for 1 and 2 issue ballots
z0 (x:y:_)    = x || y
z0 _          = True -- True for 1 issue ballots
```

Clearly, any property of ballots can be expressed like this, for in a sense we are using the built-in Boolean logic of the implementation language. We can check whether a ballot satisfies the integrity constraint the following way.

```
  > s0 [True, False, False]
False
```

We let the set of models of the integrity constraint, Mod(IC) be the set of all ballots that satisfy IC.

```
models :: Int -> IC -> [Ballot]
models m bc = [ b | b <- allBallots m, bc b]
```

An element of Mod(IC) is also called a *rational ballot* and a profile in which all ballots belong to Mod(IC) is a *rational profile*.

```
checkProfile :: Profile -> IC -> Bool
checkProfile profile p = all p profile
```

Example 1 (continued). Since Agent 1 does not vote the same on the first 2 issues, the profile of Example 1 is not rational for $s0$.

```
  > checkProfile exampleA s0
False
```

Clearly, the integrity constraint that expresses the situation of Example 1 is $q0$. Indeed all family members hold a rational ballot.

```
  > checkProfile exampleA q0
True
```

■

2.2 Aggregation rules

An *aggregation rule* is a method for aggregating the individual preferences of the agents into a collective decision. For example, when there is no constraint, we can decide on every issue according to majority. Formally we use the following definition.

Definition 2. An aggregation rule $F : \{0,1\}^{m\times n} \to \mathcal{P}(\{\mathsf{True}, \mathsf{False}\}^m)$ is a function that maps every profile to a nonempty set of ballots.

In our implementation we also take the number of issues and the integrity constraint as parameters to enable us to write more generic aggregation rules that can be used for different m, IC.

```
type AR = Int -> IC -> Profile -> [Ballot]
```

The type for the aggregation rule is such that we do not necessarily reach a unique decision. In some rules there might be multiple decisions that are equally "good" according to the rule. In that case

the agents have to agree on a way of breaking ties. Simply rolling a die is a fair option. We call an aggregation rule *resolute*, when it always provides us with exactly one outcome, that is, $|F(\mathbf{B})| = 1$ for all $\mathbf{B} \in \{0,1\}^{m \times n}$.

We create a new module for the implementation of some aggregation rules.[3]

```
module Rules where
import Aggregation
import MyBasics
import AgHelpFun
import Data.List
import Data.Tuple
import Data.Ord
```

The majority rule

We have seen the majority rule a few times before. When we accept an issue if and only if a majority does, we are applying the majority rule to the profile. If there is an even number of voters, we have two kinds of majorities, namely the strict and the weak majority. The following function gives the (strict or weak) majority resolution of a given profile, respectively denoted by $\mathsf{Maj}_S(\mathbf{B})$ and $\mathsf{Maj}_W(\mathbf{B})$.

```
majResolve :: Quotkind -> AR
majResolve kind m _bc profile =
  [[majority kind m i profile | i <- [0..(m-1)] ]]
```

If we do not make a choice in case of weak majorities, we obtain the rule $\mathsf{Maj}(\mathbf{B})$ encoded below. This rule is resolute for odd n.

```
majResolveAll :: AR
majResolveAll m _ profile = majResolveAll' 0 where
    majResolveAll' i =
      if i == m then [[]]
      else
        let
          ms = majority Strict m i profile
          mw = majority Weak m i profile
        in
          if mw == ms then map (mw :) (majResolveAll' (i+1)) else
             map (ms :) (majResolveAll' (i+1)) ++
             map (mw :) (majResolveAll' (i+1))
```

Example 2. Suppose four agents have to decide on three issues. There is no integrity constraint. Their preferences can be found in Table 2.2. This profile is encoded as follows.

[3]We use functions from the module AgHelpFun, which can be found in Appendix A.

Issue:	1	2	3
Agent 1	True	True	True
Agent 2	True	True	True
Agent 3	False	True	True
Agent 4	False	True	False

Table 2.2: Four agents express their preferences on three issues.

```
exampleF :: Profile
exampleF = [[True,True,True],
            [True,True,True],
            [False,False,True],
            [False,False,False]]
```

We can apply the weak, strict or general majority rule.

```
  > majResolve Strict 3 n0 exampleF
[[False,False,True]]
  > majResolve Weak 3 n0 exampleF
[[True,True,True]]
  > majResolveAll 3 n0 exampleF
[[False,False,True],[False,True,True],[True,False,True],[True,True,True]]
```

We see that the general majority rule gives us all combinations of the weak and strict resolutions for the different issues. ■

Axioms for aggregation rules

Example 1 - Cat or Dog (continued). For our example about the family purchasing pets, we obtain the following strict majority winner.

```
  > majResolve Strict 3 n0 exampleA
[[True,False,True]]
```

Thus if the family would apply the majority rule on all separate issues, according to majority, they should buy a dog and they should buy a cat, but they should not buy both. ■

Clearly the above example is contradictory. This problem arises because the majority rule is not *collectively rational* for $q0$.

Definition 3. Given an integrity constraint IC, an aggregation rule F is called *collectively rational* (CR) for IC if for all rational profiles $\mathbf{B}$, all elements of $F(\mathbf{B})$ are also rational. We also say that F *lifts* IC to the collective level.

As shown in Grandi and Endriss (2013), the class of integrity constraints that is lifted by the majority rule is exactly the class of formulas that are equivalent to a conjunction of clauses of size at most two. To illustrate this, we will prove the following weaker statement.

Theorem 1. *Majority voting is collectively rational with respect to the class of integrity constraints that are conjunctions of literals.*

Proof. Suppose IC is a conjunction of literals. This means that IC is equivalent to $p_{i_1} \wedge \ldots \wedge p_{i_k} \wedge \neg p_{i_{k+1}} \wedge \ldots \wedge \neg p_{i_m}$. If the agents all cast a rational ballot, then this must mean that they all vote in favour of issues $i_1, \ldots, i_k$ and not in favour of $i_{k+1}, \ldots, i_m$. Hence since unanimity is always a majority, by majority voting $i_1, \ldots, i_k$ will be accepted and $i_{k+1}, \ldots, i_m$ will be rejected, thus the outcome wil satisfy IC. ☺

Since any outcome of a rule should be possible, we should use a collectively rational rule. Because of this, we see that the majority rule is not always a possibility. Therefore we should look at other possible aggregation rules. For example, we could use the rule that always picks the ballot $(\mathsf{True}, \mathsf{True}, \ldots)$. However, this is not a very interesting rule, since we somehow want the rule to actually use the information given in the profile to generate a "reasonable" group decision. To formalize this notion of a reasonable rule, we study these rules using the axiomatic method. These axioms are properties that might be desirable for aggregation rules. We introduce here axioms for non-resolute aggregation rules. In the resolute case, these axioms are equivalent to the resolute axioms introduced in Lang et al. (2015). For these non-resolute axioms we first need the following definition.

Definition 4. We define the negation of elements in $\{\mathsf{True}, \mathsf{False}\}$ as $\neg\mathsf{True} = \mathsf{False}$ and $\neg\mathsf{False} = \mathsf{True}$. The negation of a set $S \subseteq \{\mathsf{True}, \mathsf{False}\}$ is given by $\neg S = \{x \in \{\mathsf{True}, \mathsf{False}\} \mid \exists y \in S \text{ such that } x = \neg y\}$.

Now we will introduce some axioms an aggregation rule F can satisfy. The axiom of unanimity states, that if all agents vote unanimously on an issue, the outcome of the rule should agree with them on this issue.

Unanimity (U). For any profile $\mathbf{B} \in \mathsf{Mod}(\mathrm{IC})^n$ and any $x \in \{\mathsf{True}, \mathsf{False}\}$, if $b_{i,j} = x$ for all i, then $F(\mathbf{B})_j = \{x\}$.

An aggregation rule satisfies Anonymity if it is symmetric with respect to agents.

Anonymity (A). For any any profile $\mathbf{B} \in \mathsf{Mod}(\mathrm{IC})^n$, and any permutation $\sigma : \mathcal{I} \to \mathcal{I}$, we have $F(B_1, \ldots, B_{n-1}) = F(B_{\sigma(1)}, \ldots, B_{\sigma(n-1)})$.

The axiom of issue-neutrality states that if two issues have exactly the same coalition in a profile, then the outcome(s) must be equal for the two issues.

Issue-Neutrality ($\mathbf{N}^{\mathcal{I}}$). For any two issues $j, j' \in \mathcal{I}$, and any profile $\mathbf{B} \in \mathsf{Mod}(\mathrm{IC})^n$, if $b_{i,j} = b_{i,j'}$ for all $i \in \mathcal{N}$, then $F(\mathbf{B})_j = F(\mathbf{B})_{j'}$.

Domain neutrality states that if the coalition of an issue is exactly the anticoalition of another issue, then the outcome(s) on the one issue should be the inverse of the outcome(s) on the other.

Domain-Neutrality $\mathbf{N}^{\mathcal{D}}$. For any two issues $j, j' \in \mathcal{I}$, and any profile $\mathbf{B} \in \mathsf{Mod}(\mathrm{IC})^n$, if $b_{i,j} = \neg b_{i,j'}$ for all $i \in \mathcal{N}$, then $F(\mathbf{B})_j = \neg(F(\mathbf{B})_j)$.

Responsiveness states that for any issues both True and False must be a possible outcome.

Responsiveness (R). For any issue $j \in \mathcal{I}$, there are profiles $\mathbf{B}, \mathbf{B}' \in \mathsf{Mod}(\mathrm{IC})^n$ such that $\mathsf{True} \in F(\mathbf{B})_j$ and $\mathsf{False} \in F(\mathbf{B}')_j$.

Independence states that the outcome on an issue should not depend on the votes on other issues. That is, if no agent changes his vote on issue j then the set of the outcomes for j should not change either.

Indepencence (I). For any issue $j \in \mathcal{I}$, and any two profiles $\mathbf{B}, \mathbf{B}' \in \mathsf{Mod}(\mathrm{IC})^n$, if $b_{i,j} = b'_{i,j}$ for all $i \in \mathcal{N}$, then $F(\mathbf{B})_j = F(\mathbf{B}')_j$.

We now introduce an axiom of weak independence, since this can simplify some proofs. The axiom of weak independence states that if only one agent changes his ballot, but she does not change her vote on issue j, then the j-vote set of the outcome should not change either. We will show that this axiom is actually equivalent to the axiom of independence.

Weak Independence (WI). For any issue $j \in \mathcal{I}$, any agent i, any $\min_i$-profile $\mathbf{B}_{-i} \in \mathsf{Mod}(\mathrm{IC})^n$ and for any pair of alternative ballots B_i, B'_i if $b_{i,j} = b'_{i,j}$, then $F(B_i, \mathbf{B}_{-i})_j = F(B'_i, \mathbf{B}_{-i})_j$.

Theorem 2. *F satisfies I if and only if it satisfies WI.*

Proof. Suppose F satisfies I. For the left to right direction, let j be an issue, i an agent, $\mathbf{B}_{-i}$ a $\min_i$-profile and B_i, B'_i a pair of alternative ballots such that $b_{i,j} = b'_{i,j}$. Since $b_{i',j} = b_{i',j}$ for all $i' \neq i \in \mathcal{N}$, we get by independence $F(B_i, \mathbf{B}_{-i})_j = F(B'_i, \mathbf{B}_{-i})_j$. Thus F satisfies weak independence. Now suppose F satisfies weak independence. Let j be an issue and let $\mathbf{B} = (B_1, \ldots, B_n)$, $\mathbf{B}' = (B'_1, \ldots, B'_n)$ be profiles such that $b_{i,j} = b'_{i,j}$ for all $i \in \mathcal{N}$. Now we will show

$$F(B'_0, \ldots, B'_{i-1}, B_i, \ldots B_{n-1})_j = F(B'_0, \ldots, B'_i, B_{i+1}, \ldots B_{n-1})_j \quad \forall i \in \mathcal{M},$$

by induction on i. For $i = 0$, this follows directly by weak independence. Suppose for some $i \in \mathcal{M}$ we have $F(B'_0, \ldots, B'_{i-1}, B_i, \ldots B_{n-1})_j = F(B'_0, \ldots, B'_i, B_{i+1}, \ldots B_{n-1})_j$. Then we can apply weak independence for agent $i+1$ to obtain $F(B'_0, \ldots, B'_i, B_{i+1}, \ldots B_{n-1})_j = F(B'_1, \ldots, B'_{i+1}, B_{i+2}, \ldots B_n)_j$. Hence by induction $F(B'_0, \ldots, B'_{i-1}, B_i, \ldots B_{n-1})_j = F(B'_0, \ldots, B'_i, B_{i+1}, \ldots B_{n-1})_j$ for all $i \in \mathcal{M}$. This gives us $F(\mathbf{B})_j = F(\mathbf{B}')_j$. Hence F satisfies independence. ☺

The axiom of monotonicity (which is also applicable to non-resolute rules) states that if acceptance (resp. rejection) is a possible outcome for an issue, and this issue gets more (resp. less) support, acceptance (resp. rejection) should still be a possible outcome.

Monotonicity (M)[4]**.** For any issue $j \in \mathcal{I}$, any agent $i \in \mathcal{N}$, any $\min_i$-profile $\mathbf{B}_{-i} \in \mathsf{Mod}(\mathrm{IC})^n$ and for any pair of alternative ballots B_i, B'_i such that $b_{i,j} = \neg x$ and $b'_{i,j} = x$, we have $x \in F(B_i, \mathbf{B}_{-i})_j$ implies $x \in F(B'_i, \mathbf{B}_{-i})_j$.

Monotonicity implies the following notion of weak monotonicity.

Weak Monotonicity (WM). For any issue $j \in \mathcal{I}$, any agent $i \in \mathcal{N}$ and any $\min_i$-profile $\mathbf{B}_{-i} \in \mathsf{Mod}(\mathrm{IC})^n$, if there exists a pair of ballots B_i,B'_i such that $b_{i,j} = \neg x$ and $b'_{i,j} = x$, then for some such pair, $x \in F(B_i, \mathbf{B}_{-i})_j$ implies $x \in F(B'_i, \mathbf{B}_{-i})_j$.

Note that if F is resolute and satisfies both I and WM, it also satisfies M.

For the next axiom we first need some notation. If $\mathbf{B} = (B_0, \ldots, B_{n-1})$ and $\mathbf{B}' = (B'_0, \ldots, B'_{n-1})$, let the *support* of a profile, be the set of ballots that occur at least once in the profile, that is $\mathsf{Supp}(\mathbf{B}) = \{B_0, \ldots, B_{n-1}\}$ and let $\mathbf{B} \oplus \mathbf{B}' = (B_0, \ldots, B_{n-1}, B'_0, \ldots, B'_{n-1})$.

[4]The resolute version of this definition of monotonicity was introduced by Grandi and Endriss (2011a).

The axiom of reinforcement states that if you let two separate groups (with the same support) vote, if they have outcomes in common, as a joint group the rule should produce exactly these outcomes.

Reinforcement. For any two profiles $\mathbf{B} = (B_0, \ldots, B_{n-1})$ and $\mathbf{B}' = (B'_0, \ldots, B'_{n-1})$ such that $\mathsf{Supp}(\mathbf{B}) = \mathsf{Supp}(\mathbf{B}')$ and $F(\mathbf{B}) \cap F(\mathbf{B}') \neq \emptyset$, we have $F(\mathbf{B} \oplus \mathbf{B}') = F(\mathbf{B}) \cap F(\mathbf{B}')$.

As some impossibility theorems show, for several integrity constraints a lot of these axioms cannot be combined in one collectively rational aggregation rule. Therefore, when picking a rule, you have to decide which axioms are most important to you.

We also introduce another very nice property we would like aggregation rules to have, namely that it agrees with the majority rule when possible. That is, when the majority rule provides rational outcomes, the aggregation rule should have exactly these outcomes. As the majority rule is seen as an attractive or even perfect rule, this is very appealing. This was introduced in Lang et al. (2015)

Definition 5. Let F an aggregation rule. F is *majority preserving* when for any profile $\mathbf{B}$ such that $\mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC}) \neq \emptyset$, we have $F(\mathbf{B}) = \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$.

We will introduce a number of aggregation rules and we will investigate which axioms and other properties are satisfied by these rules.

The priority rule

As suggested in List and Pettit (2002), one way to ensure collective rationality is by prioritizing issues. The procedure that is most often used for this is the *premise-based procedure*, in which certain issues are treated as premises, on which is decided by majority, where the decisions on the other issues should follow logically from the premises. To ensure this procedure is collectively rational and well-defined, the premises must be carefully chosen and for any assignment of the premises, the truth value of each of the conclusions has to be fully determined. However, this might not always be possible for every integrity constraint. Consider integrity constraint $\mathrm{IC} = p_0 \vee p_1 \vee p_2$. The majority rule does not lift IC. However, no matter which set of premises we choose, if we vote True for one of the premises, the conclusion(s) is(/are) not determined. Therefore here we suggest a more general approach following the ideas of List (2002a).

In this priority rule we assume issues are ordered by decreasing importance. We resolve the issues in order, choosing the majority outcome if this does not clash with the integrity constraint. If it does clash, we decide against the majority on the issue.

Definition 6. The priority rule is the aggregation rule given by $\mathsf{PR}(\mathbf{B}) = \mathsf{pr}^m(\mathbf{B})$, where we define $\mathsf{pr}^j(\mathbf{B})$ inductively on $j \in \{0, \ldots, m\}$ as follows.

$$\mathsf{pr}^0(\mathbf{B}) = \mathsf{Mod}(\mathrm{IC})$$

$$\mathsf{pr}^{j+1}(\mathbf{B}) = \begin{cases} \{B \in \mathsf{pr}^j(\mathbf{B}) \mid \mathsf{Maj}(\mathbf{B})_j = \{b_j\}\} & \text{if this set is nonempty ,} \\ \mathsf{pr}^j(\mathbf{B}) & \text{otherwise.} \end{cases}$$

We implement this rule.

```
priority :: AR
priority m bc profile =
```

```
let
  mjS = head (majResolve Strict m n0 profile)
  mjW = head (majResolve Weak m n0 profile)
  priority' i outcomes =
    if i == m then outcomes
    else
      let
        newOutcomes = filter (\x -> agree x mjS i || agree x mjW i)
                      outcomes
      in
        if null newOutcomes then priority' (i+1) outcomes
        else priority' (i+1) newOutcomes
in
  priority' 0 (models m bc)
```

For Example 1, if the priority order is $0, 1, 2$, we can use this rule to reach a decision. In this way the decision for a cat will depend upon the first two majority outcomes.

```
  > priority 3 q0 exampleA
[[True,False,False]]
```

With these priorities, the family should only buy a dog, but not a cat. We now study which axioms and properties the priority rule satisfies.

Theorem 3. *The priority rule is a collectively rational, well-defined aggregation rule.*

Proof. This follows quite straightforward by the definition of $\mathsf{pr}^m(\mathbf{B})$. ☺

Theorem 4. *On any profile where* $|\mathsf{Maj}(\mathbf{B})| = 1$*, we have* $|\mathsf{PR}(\mathbf{B})| = 1$.

Proof. This follows quite straightforward by the definition of $\mathsf{pr}^m(\mathbf{B})$. ☺

This implies the following corollary.

Corollary 1. *For odd* n*, the priority rule is resolute.*

We will prove the priority rule satisfies reinforcement. For this we need the following Lemma.

Lemma 1. *For any profile* $\mathbf{B}$*,* $j \in \{0, \ldots, m-1\}$*, we have* $\mathsf{pr}^{j+1}(\mathbf{B}) \subseteq \mathsf{pr}^j(\mathbf{B})$.

Proof. This follows directly from the definition of $\mathsf{pr}^m(\mathbf{B})$. ☺

Theorem 5. *The priority rule satisfies reinforcement.*

Proof. Let $\mathbf{B} = (B_0, \ldots, B_{n-1})$ and $\mathbf{B}' = (B'_0, \ldots, B'_{n-1})$ such that $\mathsf{Supp}(\mathbf{B}) = \mathsf{Supp}(\mathbf{B}')$ and $\mathsf{PR}(\mathbf{B}) \cap \mathsf{PR}(\mathbf{B}') \neq \emptyset$. This means there must be a $\tilde{B} \in \mathsf{PR}(\mathbf{B}) \cap \mathsf{PR}(\mathbf{B}')$. First note that this implies $\tilde{B} \in \mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}')$ for any $j \leq m$ by Lemma 1.

We will prove $\mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}') = \mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}')$ by induction on j. For $j = 0$ we have $\mathsf{pr}^0(\mathbf{B} \oplus \mathbf{B}') = \mathsf{Mod}(\mathrm{IC}) = \mathsf{pr}^0(\mathbf{B}) = \mathsf{pr}^0(\mathbf{B}') = \mathsf{pr}^0(\mathbf{B}) \cap \mathsf{pr}^0(\mathbf{B}')$.

Suppose for j we have $\mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}') = \mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}')$. This is our induction hypothesis, which we will continuously throughout the proof. Now we distinguish two cases

1. Suppose for all $B \in \mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}')$, we have that $b_j = \tilde{b}_j$. Then by definition of pr, $\mathsf{pr}^{j+1}(\mathbf{B} \oplus \mathbf{B}') = \mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}')$. Also, since $\tilde{B}$ is in both $\mathsf{pr}^{j+1}(\mathbf{B})$ and in $\mathsf{pr}^{j+1}(\mathbf{B}')$, we know that $\{B \in \mathsf{pr}^j(\mathbf{B}) \mid b_j = \tilde{b}_j\} \subseteq \mathsf{pr}^{j+1}(\mathbf{B})$ and similarly for $\mathbf{B}'$. Hence since by induction hypothesis for all $B \in \mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}')$ we have $b_j = \tilde{b}_j$, we get $\mathsf{pr}^j(\mathbf{B}) = \{B \in \mathsf{pr}^j(\mathbf{B}) \mid b_j = \tilde{b}_j\}$ and similarly for $\mathbf{B}'$, thus we obtain $\mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}') \subseteq \mathsf{pr}^{j+1}(\mathbf{B}) \cap \mathsf{pr}^{j+1}(\mathbf{B}')$. Hence we get, by Lemma 1 and by induction hypothesis that $\mathsf{pr}^{j+1}(\mathbf{B}) \cap \mathsf{pr}^{j+1}(\mathbf{B}') = \mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}') = \mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}') = \mathsf{pr}^{j+1}(\mathbf{B} \oplus \mathbf{B}')$.

2. If we are not in the case above, then there is a $B \in \mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}') = \mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}')$, such that $b_j \neq \tilde{b}_j$. Then both this B and $\tilde{B}$ are in both $\mathsf{pr}^j(\mathbf{B})$ and $\mathsf{pr}^j(\mathbf{B}')$, thus also in $\mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}')$. Note that $\tilde{B} \in \mathsf{pr}^{j+1}(\mathbf{B})$ and $\tilde{B} \in \mathsf{pr}^{j+1}(\mathbf{B}')$, hence we must have $\tilde{b}_j \in \mathsf{Maj}(\mathbf{B})_j$ and $\tilde{b}_j \in \mathsf{Maj}(\mathbf{B}')_j$.

   Now we must be in one of the following three cases. Either $\mathsf{Maj}(\mathbf{B})_j = \{\tilde{b}_j\}$ and $\mathsf{Maj}(\mathbf{B}')_j = \{\tilde{b}_j\}$, in which case $\mathsf{Maj}(\mathbf{B} \oplus \mathbf{B}')_j = \{\tilde{b}_j\}$, hence $\mathsf{pr}^{j+1}(\mathbf{B} \oplus \mathbf{B}') = \{B \in \mathsf{pr}^j(\mathbf{B}') \cap \mathsf{pr}^j(\mathbf{B}) \mid b_j = \tilde{b}_j\} = \{B \in \mathsf{pr}^j(\mathbf{B}) \mid b_j = \tilde{b}_j\} \cap \{B \in \mathsf{pr}^j(\mathbf{B}') \mid b_j = \tilde{b}_j\} = \mathsf{pr}^{j+1}(\mathbf{B}) \cap \mathsf{pr}^{j+1}(\mathbf{B}')$.

   Or we have that w.l.o.g. $\mathsf{Maj}(\mathbf{B})_j = \{\tilde{b}_j, b_j\}$ and $\mathsf{Maj}(\mathbf{B}')_j = \{\tilde{b}_j\}$. This implies $\mathsf{Maj}(\mathbf{B} \oplus \mathbf{B}')_j = \{\tilde{b}_j\}$. Thus we get $\mathsf{pr}^{j+1}(\mathbf{B}) \cap \mathsf{pr}^{j+1}(\mathbf{B}') = \mathsf{pr}^j(\mathbf{B}) \cap \{B \in \mathsf{pr}^j(\mathbf{B}') \mid b_j = \tilde{b}_j\} = \{B \in \mathsf{pr}^j(\mathbf{B}') \cap \mathsf{pr}^j(\mathbf{B}) \mid b_j = \tilde{b}_j\} = \mathsf{pr}^{j+1}(\mathbf{B} \oplus \mathbf{B}')$.

   Or we have that $\mathsf{Maj}(\mathbf{B})_j = \{\tilde{b}_j, b_j\}$ and $\mathsf{Maj}(\mathbf{B}')_j = \{\tilde{b}_j, b_j\}$, in which case $\mathsf{Maj}(\mathbf{B} \oplus \mathbf{B}')_j = \{\tilde{b}_j, b_j\}$, hence $\mathsf{pr}^{j+1}(\mathbf{B}) \cap \mathsf{pr}^{j+1}(\mathbf{B}') = \mathsf{pr}^j(\mathbf{B}) \cap \mathsf{pr}^j(\mathbf{B}') = \mathsf{pr}^j(\mathbf{B} \oplus \mathbf{B}') = \mathsf{pr}^{j+1}(\mathbf{B} \oplus \mathbf{B}')$, by induction hypothesis.

Hence the priority rule satisfies reinforcement. ☺

Furthermore we can easily see that the priority strategy satisfies anonymity. However depending on the integrity constraint and the ordering, it might violate unanimity, neutrality, independence and monotonicity. We give an example where the priority rule violates unanimity.

Example 3. Suppose $n = 3$, $m = 4$ and consider the integrity constraint $\mathrm{IC} = (p_0 \wedge p_1 \wedge p_2) \to \neg p_3$ and the profile given in Table 2.3. Clearly all agents satisfy the integrity constraint.

Issue:	0	1	2	3
Agent 0	$\top$	$\top$	$\bot$	$\top$
Agent 1	$\bot$	$\top$	$\top$	$\top$
Agent 2	$\top$	$\bot$	$\top$	$\top$

Table 2.3: 3 agents vote on 4 issues.

```
exEic :: IC
exEic (x0:x1:x2:x3:_) = (x0 && x1 && x2) --> not x3
exEic _ = True

exampleE :: Profile
```

```
exampleE = [[True,True,False,True],[False,True,True,True],
            [True,False,True,True]]
```

We now show that the profile satisfies the integrity constraint and does not accept the unanimous vote on issue 3.

```
  > checkProfile exampleE exEic
True
  > priority 4 exEic exampleE
[[True,True,True,False]]
```

This shows that the priority rule is not for all integrity constraints a unanimous rule. ■

The Priority rule does satisfy another very nice property.

Theorem 6. *The priority rule is Majority preserving.*

Proof. Let PR denote the priority rule, and Maj denote the general majority rule. Let $\mathbf{B}$ be such that $\mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC}) \neq \emptyset$. So there is a $\tilde{B} \in \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$. Take now an arbitrary $B \in \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$. We prove $B \in PR(\mathbf{B})$ by proving $B \in \mathsf{pr}^j(\mathbf{B})$ by induction on j. We have $B \in \mathsf{Mod}(\mathrm{IC}) = \mathsf{pr}^0(\mathbf{B})$.

Now suppose $B \in \mathsf{pr}^j(\mathbf{B})$, then we have either $\mathsf{Maj}(\mathbf{B})_j = \{b_j\}$, since $B \in \mathsf{Maj}(\mathbf{B})$, which gives us $B \in \mathsf{pr}^{j+1}(\mathbf{B})$. Or we have $\mathsf{Maj}(\mathbf{B})_j = \{b_j, \neg b_j\}$, thus $\mathsf{pr}^{j+1}(\mathbf{B}) = \mathsf{pr}^j(\mathbf{B})$, which also gives $B \in \mathsf{pr}^{j+1}(\mathbf{B})$. Hence by induction we have $B \in \mathsf{pr}^j(\mathbf{B})$ for all j, so $B \in PR(\mathbf{B})$. Hence we have $\mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC}) \subseteq PR(\mathbf{B})$ and specifically we have $\tilde{B} \in PR(\mathbf{B})$.

Now suppose $B \notin \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$. This must mean that either $B \notin \mathsf{Mod}(\mathrm{IC})$, which clearly implies $B \notin PR(\mathbf{B})$. Or we have $B \notin \mathsf{Maj}(\mathbf{B})$. This implies that there is a j such that $b_j \notin \mathsf{Maj}(\mathbf{B})_j$. Hence we must have $\mathsf{Maj}(\mathbf{B})_j = \{\neg b_j\}$. Now since we have that $\tilde{B} \in \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC}) \subseteq PR(\mathbf{B})$, we have $\tilde{b}_j = \neg b_j$ and we have $\tilde{B} \in \mathsf{pr}^j(\mathbf{B})$ for any j, hence $\{B' \in \mathsf{pr}^j(\mathbf{B}) \mid \mathsf{Maj}(\mathbf{B})_j = \{b'_j\}\}$ is nonempty, hence $\mathsf{pr}^{j+1}(\mathbf{B}) = \{B' \in \mathsf{pr}^j(\mathbf{B}) \mid \mathsf{Maj}(\mathbf{B})_j = \{b'_j\}\}$, hence $B \notin \mathsf{pr}^{j+1}(\mathbf{B})$. This shows $PR(\mathbf{B}) \subseteq \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$.

The two above together give us $\mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC}) = PR(\mathbf{B})$, which was what we wanted to prove. ☺

Distance-based rule

A measure for the difference between two ballots B, B' is the Hamming distance

$$\mathcal{H}(B, B) := |\{j \mid B_j \neq B'_j\}|,$$

which is the number of issues on which the ballots disagree. We can encode this

```
hamming :: Ballot -> Ballot -> Int
hamming i j = hamming' i j 0 where
```

```
hamming' [] _ d = d
hamming' _ [] d = d
hamming' (x:xs) (y:ys) d
  | x == y = hamming' xs ys d
  | otherwise = hamming' xs ys (d + 1)
```

The Hamming distance between a ballot B and a profile $\mathbf{B}$ is the sum of the Hamming distances between B and the ballots in $\mathbf{B}$.

$$\mathcal{H}(\mathbf{B}, B) := \sum_{B' \in \mathbf{B}} \mathcal{H}(B, B')$$

Since an outcome of a rule should represent the opinions of the agents and thus agree with them as much as possible, a logical choice for a rule is the following distance-based rule, as introduced in Pigozzi (2006) and Miller and Osherson (2008).

Definition 7. The *distance-based rule* DB is the aggregation rule that selects from the ballots that agree with the integrity constraint, the ballots that minimise the Hamming distance to the profile. That is,

$$\mathsf{DB}(\mathbf{B}) = \operatorname*{arg\,min}_{B \in \mathsf{Mod}(\mathrm{IC})} \mathcal{H}(\mathbf{B}, B).$$

```
dbRule :: AR
dbRule m bc profile = let
    f x = profileSum (hamming x) profile
  in
    minimaBy (comparing f) (models m bc)
```

For Example 1, we get a set of distance based outcomes.

```
  > dbRule 3 q0 exampleA
[[False,False,True],[True,False,False],[True,True,True]]
```

Clearly DB is not resolute. A tie breaking rule could be used to make a decision.

Theorem 7. *The distance-based rule satisfies reinforcement.*

Proof. Let $\mathbf{B}, \mathbf{B}'$ be such that $\mathsf{Supp}(\mathbf{B}) = \mathsf{Supp}(\mathbf{B}')$ and $\mathsf{DB}(\mathbf{B}) \cap \mathsf{DB}(\mathbf{B}') \neq \emptyset$. Let $B \in \mathsf{DB}(\mathbf{B}) \cap \mathsf{DB}(\mathbf{B}')$. Then $\mathcal{H}(B, \mathbf{B})$ and $\mathcal{H}(B, \mathbf{B}')$ are both minimal. Hence $\mathcal{H}(B, \mathbf{B} \oplus \mathbf{B}') = \mathcal{H}(B, \mathbf{B}) + \mathcal{H}(B, \mathbf{B}')$ is minimal. Hence $B \in \mathsf{DB}(\mathbf{B} \oplus \mathbf{B}')$. Now for the other direction, suppose $B \notin \mathsf{DB}(\mathbf{B}) \cap \mathsf{DB}(\mathbf{B}')$. Without loss of generality we may assume $B \notin \mathsf{DB}(\mathbf{B})$. Now since $\mathsf{DB}(\mathbf{B}) \cap \mathsf{DB}(\mathbf{B}') \neq \emptyset$ there is a $B' \in \mathsf{DB}(\mathbf{B}) \cap \mathsf{DB}(\mathbf{B}')$. For B' we must have $\mathcal{H}(B', \mathbf{B}) < \mathcal{H}(B, \mathbf{B})$ and $\mathcal{H}(B', \mathbf{B}') \leq \mathcal{H}(B, \mathbf{B}')$. Thus we have $\mathcal{H}(B', \mathbf{B} \oplus \mathbf{B}') = \mathcal{H}(B', \mathbf{B}) + \mathcal{H}(B', \mathbf{B}') < \mathcal{H}(B, \mathbf{B}) + \mathcal{H}(B, \mathbf{B}') < \mathcal{H}(B, \mathbf{B} \oplus \mathbf{B}')$. Hence $B \notin \mathsf{DB}(\mathbf{B} \oplus \mathbf{B}')$. Together this gives $\mathsf{DB}(\mathbf{B} \oplus \mathbf{B}') = \mathsf{DB}(\mathbf{B}) \cap \mathsf{DB}(\mathbf{B}')$. ☺

Furthermore, it clearly satisfies anonymity, but depending on the integrity constraint it might violate all other axioms.

Below we will show that the distance based rule is Majority preserving, which makes it an appealing rule.

Theorem 8. *The distance based rule is Majority preserving.*

Proof. Let DB denote the distance based rule, and Maj denote the general majority rule. Let $\mathbf{B}$ be such that $\mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC}) \neq \emptyset$. Thus we have some $\tilde{B} \in \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$. Now suppose we have an arbitrary $B \in \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$. We have for any ballot B', any issue j that $|\{B'' \in \mathbf{B} \mid B'_j = B''_j\}| \leq |\{B'' \in \mathbf{B} \mid B''_j = B_j\}|$. Hence for the Hamming distance we get

$$\begin{aligned}\mathcal{H}(\mathbf{B}, B') &= \sum_j n - |\{B'' \in \mathbf{B} \mid B'_j = B''_j\}| \geq \\ &\sum_j n - |\{B'' \in \mathbf{B} \mid B''_j = B_j\}| \\ &= \mathcal{H}(\mathbf{B}, B)\end{aligned}$$

Hence $\mathcal{H}(\mathbf{B}, B') \geq \mathcal{H}(\mathbf{B}, B)$ for any $B' \in \mathsf{Mod}(\mathrm{IC})$, thus $B \in \mathsf{DB}(\mathbf{B})$. Hence we have that $\mathsf{Maj}(\mathbf{B}) \subseteq \mathsf{DB}(\mathbf{B})$.

Now suppose $B \notin \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$. If $B \notin \mathsf{Mod}(\mathrm{IC})$, then $B \notin \mathsf{DB}(\mathbf{B})$. If $B \notin \mathsf{Maj}(\mathbf{B})$, then we have that there is a j such that $|\{B'' \in \mathbf{B} \mid \neg b_j = b''_j\}| > |\{B'' \in \mathbf{B} \mid b''_j = b_j\}|$. Now consider $\tilde{B} \in \mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC})$ (which exists since $\mathsf{Maj}(\mathbf{B}) \cap \mathsf{Mod}(\mathrm{IC}) \neq \emptyset$). Then we must have $\tilde{b_j} = \neg b_j$, hence $|\{B'' \in \mathbf{B} \mid \tilde{b}_j = b''_j\}| > |\{B'' \in \mathbf{B} \mid b''_j = b_j\}|$. Also for all $k \neq j$, $|\{B'' \in \mathbf{B} \mid \tilde{b}_k = b''_k\}| \geq |\{B'' \in \mathbf{B} \mid b''_k = b_k\}|$. Now we get

$$\begin{aligned}\mathcal{H}(\mathbf{B}, B) &= \sum_l n - |\{B'' \in \mathbf{B} \mid b_l = b''_l\}| > \\ &\sum_l n - |\{B'' \in \mathbf{B} \mid b''_l = \tilde{b}_l\}| \\ &= \mathcal{H}(\mathbf{B}, \tilde{B})\end{aligned}$$

Hence $\mathcal{H}(\mathbf{B}, B) > \mathcal{H}(\mathbf{B}, \tilde{B})$, thus $B \notin \mathsf{DB}(\mathbf{B})$. From this we can conclude $\mathsf{Maj}(\mathbf{B}) = \mathsf{DB}(\mathbf{B})$. ☺

Minimax rule

In a family (or within another group) it might be more important to make sure that nobody is really unhappy than to make some peoply really happy. We suggest a way to prevent depressed family members: minimise the maximal Hamming distance to the individual ballots of the agents. We call this the minimax rule, which resembles the minimax rule for approval voting given in Brams et al. (2007). This rule was also suggested in Lang et al. (2011).

Definition 8. The minimax rule MM is the aggregation rule that selects from all rational ballots, those ballots that minimise the maximal Hamming distance to the individual ballots in the profile. That is,

$$\mathsf{MM}(\mathbf{B}) = \underset{B \in \mathsf{Mod}(\mathrm{IC})}{\arg\min} \max_{i \in \mathcal{N}} \mathcal{H}(B_i, B).$$

We implement this rule.

```
mmRule :: AR
mmRule m bc profile = let
    bs = models m bc
    maxdist b = maximum (map (hamming b) profile)
  in
    minimaBy (comparing maxdist) bs
```

We can apply this rule to Example 1 and we see there are three outcomes that minimise this maximal distance.

```
  > mmRule 3 q0 exampleA
[[True,True,True],[True,False,False],[False,False,True]]
```

Example 4 - Culture or Nature. Suppose you are making a group travel with five persons and every day you have to choose between a trip in nature and a cultural trip. Three of the five persons always prefer a cultural trip, one likes culture a lot, but would like to make a trip in nature on the last day and one person wants to make a trip in nature every day. The total journey takes 5 days. Let $m = 5$ and let issue i denote the statement "we make a cultural trip on day $i + 1$". Thus the profile is given as follows.

```
exampleProfTravel :: Profile
exampleProfTravel = [[True,True,True,True,True],
                     [True,True,True,True,True],
                     [True,True,True,True,True],
                     [True,True,True,True,False],
                     [False,False,False,False,False]]
```

Every day you can do watever you like, hence there is no integrity constraint. Deciding by majority, you end up making a cultural trip every day.

```
  > majResolveAll 5 n0 exampleProfTravel
[[True,True,True,True,True]]
```

However, it seems somewhat unfair to never do what the fifth group-member wants. We could also decide to apply the minimax rule. This would give the following outcome.

```
  > mmRule 5 n0 exampleProfTravel
[[True,True,True,False,False],
[True,True,False,True,False],
[True,True,False,False,True],
[True,True,False,False,False],
[True,False,True,True,False],
[True,False,True,False,True],
[True,False,True,False,False],
[True,False,False,True,True],
[True,False,False,True,False],
```

```
[False,True,True,True,False],
[False,True,True,False,True],
[False,True,True,False,False],
[False,True,False,True,True],
[False,True,False,True,False],
[False,False,True,True,True],
[False,False,True,True,False]]
```

In this way, the opinions of the minorities are also taken into account. We could combine this rule with for example the distance based rule to choose from this large number of possibilities. ■

This example shows that the minimax rule is not majority preserving. However, the minimax rule does have another nice property.

Theorem 9. *The minimax rule satisfies reinforcement.*

Proof. To see this, note that $MM(\mathbf{B}) = \arg\min_{B \in \mathsf{Mod(IC)}} \max_{B' \in \mathsf{Supp}(\mathbf{B})} \mathcal{H}(B', B)$. Now reinforcement follows trivially. ☺

In Example 4, the anomalous preferences of agent 4 have a big influence on the outcome in the minimax rule. This might seem somewhat unfair. Therefore we suggest a rule that is somehow in between the minimax and the distance based rule, namely the least squares rule. We will see that it gives an intuitively more satisfactory outcome for this example.

Least squares rule

Definition 9. The least squares rule LS is the aggregation rule that selects from all rational ballots those ballots that minimise the sum of the squared Hamming distance to ballots from the profile. That is,

$$\mathsf{LS}(\mathbf{B}) = \operatorname*{arg\,min}_{B \in \mathsf{Mod(IC)}} \sum_{i=0}^{n-1} (\mathcal{H}(B_i, B))^2.$$

In this rule, the votes of outliers are more important to the decision than the more "average" votes.

```haskell
lsRule :: AR
lsRule m bc profile = let
    sq x y = hamming x y ^ (2 :: Int)
    f x = profileSum (sq x) profile
  in
    minimaBy (comparing f) (models m bc)
```

Theorem 10. *The least squares rule satisfies reinforcement.*

Proof. This is proven similarly to Theorem 7. ☺

In Example 1, the least squares rule gives the same outcomes as the minimax rule.

```
  > mmRule 3 q0 exampleA
[[True,True,True],[True,False,False],[False,False,True]]
```

Example 4 - Culture or Nature (continued). We will now apply the least squares rule to Example 4.

```
  > lsRule 5 n0 exampleProfTravel
[[True,True,True,True,False]]
```

We see that this rule provides us with the intuitive outcome of making a trip to nature one day, and making cultural trips the other days. This seems like an optimal solution in between the minimax and the distance based rule. ■

Note that this is again an example that shows that the least squares rule is not majority preserving. One could take this example to show that being majority preserving is not always a desirable property for an aggregation rule.

2.3 Preference aggregation

Up to now we studied decision making on binary issues. We will now study judgement aggregation, in which agents have to express their preferences over a set of alternatives. To illustrate this, we start with an example.

Example 5 - Painting a house. Suppose three family members have to decide which color to paint their house. The paint store only sells blue, red and yellow paint, so they have three options. Let the corresponding alternatives be 0=blue, 1=red, 2=yellow. Suppose the family members have the following preferences over the alternatives.

$$\text{Agent 0: } 0 > 1 > 2,$$
$$\text{Agent 1: } 1 > 0 > 2,$$
$$\text{Agent 2: } 2 > 1 > 0.$$

How should the family decide which color to paint their house? ■

Basic definitions

In preference aggregation, agents have to express their preferences over a finite set of alternatives $\chi = 0, 1, \ldots, v-1$, by giving a linear ordering on χ.

```
type Alternative = Int
type Preference = [Alternative]
```

We can translate the problem of preference aggregation into our framework for binary judgement aggregation (Grandi and Endriss (2011a)). Every pair of alternatives (i, j) will represent an issue,

namely whether or not i is preferred over j. Thus if I vote True for (i, j), that means I prefer i over j. We map pairs of alternatives to issues and back.

```
alt2iss :: (Alternative, Alternative) -> Int -> Issue
alt2iss (i,j) v = j + i * v

iss2alt :: Issue -> Int -> (Alternative,Alternative)
iss2alt i v = (div i v,rem i v)
```

In Example 5, where $\chi = \{0, 1, 2\}$, we can translate the preference blue>red, which corresponds to the pair $(0, 1)$ to the issue numbered 1.

```
  > alt2iss (0,1) 3
1
  > iss2alt 1 3
(0,1)
```

We can also translate a preference order to a ballot and back.

```
pref2ballot :: Preference -> Ballot
pref2ballot pref = let
    v = length pref
    p2b i | i == v * v = []
          | elemIndex (fst (iss2alt i v)) pref
              < elemIndex (snd (iss2alt i v)) pref = True : p2b (i+1)
          | otherwise = False : p2b (i+1)
  in
    p2b 0

ballot2pref :: Int -> Ballot -> Preference
ballot2pref v ballot = let
    f i j | i == j = EQ
          | ballot !! alt2iss (i,j) v = LT
          | otherwise = GT
  in
    sortBy f [0..(v-1)]

ballots2prefs :: Int -> [Ballot] -> [Preference]
ballots2prefs v = map (ballot2pref v)

pref2int :: Preference -> BalInt
pref2int = b2i . pref2ballot

int2pref :: Int -> BalInt -> Preference
int2pref v = ballot2pref v . i2b (v * v)
```

```
  > pref2ballot [0,1,2]
[False,True,True,False,False,True,False,False,False]
```

Similarly we can translate a complete profile, which consist of a preference order for every agent.

```
prefs2profile :: [Preference] -> ProfInt
prefs2profile = map pref2int

profile2prefs :: Int -> ProfInt  -> [Preference]
profile2prefs v = map (int2pref v)
```

Example 5 - Painting a house (continued). The ballots corresponding to the preference profile of our family can be found in Table 2.4. Translated into integers, our profile is given by `[200,104,38]`

Pair:	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)
Issue:	0	1	2	3	4	5	6	7	8
Member 0	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
Member 1	$\bot$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
Member 2	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\bot$

Table 2.4: Three family members express their preferences for a color for their house. The preference profile is translated into a binary judgement profile. To fit in the Table, $\top$ is shorthand for True, $\bot$ for False.

```
exampleD :: Profile
exampleD = is2bs 9 [200,104,38]
```

We can check this.

```
  > profile2prefs 3 exampleD
[[0,1,2],[1,0,2],[2,1,0]]
```

■

Clearly with this translation we obtain $m = v^2$ issues. We will extend our language $\mathcal{L}_{PS}$, with symbols $p_{(a,b)}$ for the issues that correspond to the pairs (a, b). Our binary preferences should represent a linear ordering on alternatives. This is captured by the following set of integrity constraints, which we call $\text{IC}_<$:

Antisymmetry:	$p_{(a,b)} \leftrightarrow \neg p_{(b,a)}$	for $a \neq b \in \chi$
Irreflexivity:	$\neg p_{(a,a)}$	for all $a \in \chi$
Transitivity:	$p_{(a,b)} \wedge p_{(b,c)} \rightarrow p_{(a,c)}$	for $a, b, c \in \chi$ pairwise distinct

We can encode this as `lo0 v`.

```
lo0 :: Int -> IC
lo0 v ballot = let
    diag = [(a,a) | a <- [0..(v-1)]]
    diagelems = [ ballot!!i | i <- [0..(v * v - 1)],
```

```
                iss2alt i v `elem` diag]
    irrefl = all (== False) diagelems
    antisymPart i =  (iss2alt i v `elem` diag) ||
      (ballot !! i /= ballot !! alt2iss (swap (iss2alt i v)) v)
    antisym = all antisymPart [0..(v * v - 1)]
    tr i x = (ballot !!i && ballot !! x
           && fst (iss2alt x v) == snd (iss2alt i v)) -->
         ballot !! alt2iss (fst (iss2alt i v),snd (iss2alt x v)) v
    transPart i = all (tr i) [0..(v*v-1)]
    transitive = all transPart [0..(v * v - 1)]
  in
    irrefl && antisym && transitive
```

Because we have this translation, we can now also use our implementation to tackle problems in preference aggregation. Or maybe better: our implementation is an implementation of basic concepts in preference aggregation. Though not the most efficient one, because of the exponential blowup and the necessary translations. We can now implement and use rules for preference aggregation, and we can even apply some of the rules we already have, such as the distance based rule. Any aggregation function that is collectively rational for $\mathrm{IC}_<$ coincides with a social welfare function, which is a function returning a (set of) preference ordering(s). However, when choosing from alternatives, we are often not specifically interested in a collective ordering of the Alternatives, but more in one winner or a set of winners. For this we use social choice functions, which are defined as follows.

Definition 10. A *social choice function* is a function $F : (\mathsf{Mod}(\mathrm{IC}_<))^{\mathcal{N}} \to \mathcal{P}(\chi)$, selecting one or more of the alternatives, given a preference profile.

```
type SCF = Int -> Profile -> [Alternative]
```

We call a social choice function *resolute* if $|F(\mathbf{B})| = 1$ for all $\mathbf{B} \in \mathsf{Mod}(\mathrm{IC}_<)$.

When deciding on a winner from a preference profile, an appealing way to compare alternatives is by pairwise majority comparison. We let y be a pairwise majority winner against x (we write $y >^{\mu} x$), when $\{B \in \mathbf{B} \mid B_{(y,x)} = \mathsf{True}\} > \{B \in \mathbf{B} \mid B_{(y,x)} = \mathsf{False}\}$. Intuitively a winner should also win as much pairwise majority comparisons as possible. This gives rise to the following social choice function suggested by Copeland (1951).

The Copeland rule

The Copeland rule uses the *Copeland score* of an alternative x, given a certain profile, which is as follows

$$\mathsf{Copeland}(x) = |\{y \in \chi \mid x >^{\mu} y\}| - |\{y \in \chi \mid y >^{\mu} x\}|,$$

Thus the Copeland score of an alternative is the number of alternatives it would defeat in a pairwise majority elecation, minus the number of alternatives it would lose against. For example, if we look at Example 5, we see that blue wins against yellow in a pairwise majority comparison, but it loses against red. Hence its Copeland score is 0.

We encode a function that calculates the Copeland score of an alternative given a profile.

```
copelandScore :: Alternative -> Int -> Profile -> Int
copelandScore i v profile = let
    m = v * v
    iFirst = [alt2iss (i,b) v | b <- [0..(v-1)]]
    iSnd = [alt2iss (b,i) v | b <- [0..(v-1)]]
  in
    length [j | j <- iFirst, majority Strict m j profile] -
    length [j | j <- iSnd, majority Strict m j profile]
```

Note that this function uses the translation from alternatives to issues.

Indeed the Copeland score of blue in example 5 is 0.

```
  > copelandScore 0 3 exampleD
0
```

Definition 11. The Copeland rule Copeland is the social choice function that selects from ξ those alternatives with the highest Copeland score.

We implement the Copeland rule.

```
copelandRule :: SCF
copelandRule v profile =
  maximaBy (comparing (\i -> copelandScore i v profile)) [0..(v-1)]
```

The Copeland winner of Example 5 is alternative 1.

```
  > copelandRule 3 exampleD
[1]
```

Thus according to the Copeland rule, the family should paint their house red.

We could study social choice functions in the same fashion as we studied binary aggregation rules, by characterizing them in terms of the axioms they satisfy, as is done in Zwicker (2016). This is outside the scope of this thesis.

Summary

We now have an implementation of all the basic concepts in binary judgement aggregation and preference aggregation. We also implemented several aggregation rules and a social choice function. Some more aggregation rules can be found in appendix A.3. This implementation can also be used to implement any rule of your own.

In the next chapter, we turn our attention to manipulation of the decision making process, in which we will use our implementation of the priority and the distance based rule. We will also return to social choice functions and explain how these can be manipulated.

3 | Manipulating decision making

In this chapter, we will discuss several ways in which an agent could influence the process of decision making such that the final decision will (probably) be more benificial to her. There are several ways in which an agent can manipulate this process. We will start with two examples in which an agent can ensure a better outcome for herself by lying about her preferences.

```
module Manipulation where
import Aggregation
import AgHelpFun
import MyBasics
import Rules
import Data.Ord (comparing)
import Data.List
import Text.Read (readMaybe)
import Control.Arrow
```

Example 6 - Spending money. Consider a family of three members that has to decide whether they want to buy a new kitchen, buy a car, go on holidays, remodel their garden and/or change their floor. If they buy both a new kitchen and a new car, no money is left for the other expenses. If they buy only one of the two, they have enough money for the rest. Thus they have the integrity constraint $l_0 = p_0 \wedge p_1 \rightarrow (\neg p_2 \wedge \ldots \wedge \neg p_{m-1})$. The individual preferences of the family members can be found in Table 3.1. In the implementation, this profile is given by `[15,23,8]`.

Issue:	0 : new kitchen	1 : new car	2 : vacation	3: remodel garden	4 : change floor
Member 0	False	True	True	True	True
Member 1	True	False	True	True	True
Member 2	False	True	False	False	False

Table 3.1: Three family members cast their vote on five issues.

```
exampleC :: Profile
exampleC = is2bs 5 [15,23,8]
```

Suppose the family members decide to prioritize the decisions in order $0, 1, 2, 3, 4$. They decide by majority only if the decision is not forced by the integrity constraint. If they do this they get the

following outcome.

```
  > priority 5 l0 exampleC
[[False,True,True,True,True]]
```

This result agrees on two issues with the preferences of agent 2. Now suppose she really hates going on vacation and she wants to keep the garden and floor as it is. Agent 2 could choose to misrepresent her preferences, by casting the ballot (True, True, False, False, False), which has the integer representation 24. We can apply the priority rule to the profile in which she lies.

```
  > priority 5 l0 [15,23,24]
[[True,True,False,False,False]]
```

This way, they will get a new kitchen, but the other three issues agent 2 did not agree with are not happening. The result now agrees on four issues with the preferences of agent 2. Hence, if all issues are equally important to agent 2, by misrepresenting her preferences, she managed to force an outcome that is more beneficial to her. ■

In the previous example, an agent could lie about her preferences to get her way on more issues. Another case in which it might not be optimal to cast an honest ballot, is if certain issues are more important to you than other ones. We show this with Example 1.

Example 1 - Cat or Dog (continued). Remember that the issues were numbered 0="dog", 1="cat and dog", 2="cat". Suppose that the family members decide to solve the issues by priority, where issues are prioritized in the order $0, 1, 2$. If they do this, on base of their true preferences, they will get a dog, and not a cat.

```
  > priority 3 q0 exampleA
[[True,False,False]]
```

However, if family member 1 rather has a cat than a dog, she can choose to misrepresent her preferences, by saying that she only wants a cat. If we now apply the priority strategy, the family will not decide to buy a dog, and they will not decide to buy both a cat and a dog. These decisions do not force a decision on buying a cat, hence by applying majority on this issue, they decide to buy a cat.

```
  > priority 3 q0 [4,1,1]
[[False,False,True]]
```
■

The above are examples of manipulation of the aggregation rule. When agents are provided with only partial knowledge about the preferences of the others, they try to maximize their expected "satisfaction with the outcome" by lying about their preferences. There are more ways in which an agent can try to manipulate the outcome. For example, she could influence the choice of an aggregation rule, or make sure the right issues are on the agenda. The process consisting of picking a rule, setting the agenda and the application of the aggregation rule is called the decision making process. We will first discuss manipulation of the aggregation rule by lying about ones preferences. Then we will discuss how agents can manipulate the choice of a rule and the agenda setting.

3.1 Measuring satisfaction

In literature, there are several different definitions of manipulability of aggregation rules (e.g. the definition given in Grandi (2012) is a special case of the definition in Dietrich and List (2007b)). These different definitions make some assumptions on the preferences of agents. For example, in Grandi (2012) it is assumed that agents prefer an outcome that has a smaller Hamming distance to their true ballot. In Dietrich and List (2007b), it is assumed that agents are closeness-respecting, that is if $\mathsf{Agree}(B, B') \subseteq \mathsf{Agree}(B, B'')$, where B represents the true preferences of agent i, then agent i will like B'' better or equally as B'. However, this does not nessecarily have to be the case.

Example 7. Suppose I prefer to have only a dog and no cat, but if we do not get a dog, then I would rather have a cat than no pet at all. This means that on the issues (Dog, Cat and Dog, Cat) my true preference is (True, False, False), but the outcome (False, False, True) is better for me than (False, False, False), even though $\mathsf{Agree}((\mathsf{True}, \mathsf{False}, \mathsf{False}), (\mathsf{False}, \mathsf{False}, \mathsf{True})) \subseteq \mathsf{Agree}((\mathsf{True}, \mathsf{False}, \mathsf{False}), (\mathsf{False}, \mathsf{False}, \mathsf{False}))$. ■

The example above shows that the assumption that preferences should be closeness-respecting is not straightforward and neglects a lot of cases we might come across in real life. We will therefore depart from the literature by giving a more general definition for manipulability of the aggregation rule[1], which we later give in Definition 16. For discussing manipulation under full knowledge, only an ordering on outcomes is nessecary. However, to enable us to extend our framework to manipulation under partial knowledge and to do measurements on the lucrativity of manipulation we will use the game theoretical concept of a utility function, that gives a measure of satisfaction experienced by an agent when presented with a certain outcome.

Since outcomes are sets of ballots, to represent satisfaction with the outcome, we define for each agent $i \in \mathcal{N}$, a *utility function* $\mathcal{U}_i : \mathcal{P}(\mathsf{Mod}(\mathrm{IC})) \to \mathbb{R}_{\geq 0}$ on sets of ballots. This utility function assigns to each set of rational ballots S a number, expressing the agents satisfaction with S as an outcome. We can extend this definition to a utility on single ballots, where we let $U_i(B) = U_i(\{B\})$.

```
type UtilitySingle = Ballot -> Float
type UtilitySet = [Ballot] -> Float

uSingle :: UtilitySet -> UtilitySingle
uSingle u ballot = u [ballot]
```

Definition 12. A utility function U_i is called *set-consistent* if for any set of ballots S, $U_i(S) \leq \max_{B \in S} U_i(B)$ with equality if and only if $U_i(B') = \max_{B \in S} U_i(B) \quad \forall B' \in S$.

Definition 13. A set-consistent utility function U_i is called *top-respecting* to a ballot B if $U_i(B) = \max_{B' \in \mathsf{Mod}(\mathrm{IC})} U_i(B')$.

In the remainder of the thesis we will assume any utility U_i is set-consistent and top-respecting to the ballot representing the true preferences of agent i, thus if we say U_i is the utility of an agent, we assume his true ballot is one which U_i is top respecting to.

[1] Dietrich and List (2007b) also give a definition of manipulability on a set $Y \subseteq \mathcal{I}$. For this definition, there are even more restrictions on the agent. She does not only need to be closeness-respecting, but she should also only "care" about issues in Y. That is, she should like ballots that only differ on issues in Y equally.

We define $\mathsf{Top}(U_i)$ as the set of ballots for which the utility is maximal. Thus

$$\mathsf{Top}(U_i) = \{B \in \mathsf{Mod}(\mathrm{IC}) | U_i(B) \geq U_i(B') \quad \forall B' \in \mathsf{Mod}(\mathrm{IC})\}.$$

Given a utility function we calculate the true preferences of an agent.

```
truePrefs :: Int -> UtilitySingle -> IC -> [Ballot]
truePrefs m u bc = maximaBy (comparing u) (models m bc)
```

There are other properties that utility functions might satisfy. For example a natural assumption on a utility function is the following.

Definition 14. A utility function is called *average-respecting* if the utility of a set is the average of the utilities of the elements.

If ties are broken at random, the average-respecting utility gives the expected utility of the single outcome. This shows that in some cases it is reasonable to assume utility functions are average-respecting. If we are provided with a utility only defined on single ballots, we can calculate a corresponding full utility function. We encode this.

```
uSet :: UtilitySingle  -> UtilitySet
uSet u ballots = sum (map u ballots) /
                   fromIntegral (length ballots)
```

Another property that utility functions can have is quite interesting, since this property is often assumed when studying manipulability.

Definition 15. We call a utility function U_i *closeness-respecting* to a ballot B if for any B', B'', we have that $\mathsf{Agree}(B, B') \subseteq \mathsf{Agree}(B, B'')$ implies $U_i(B') \leq U_i(B'')$. A utility function is called closeness-respecting if it is closeness-respecting to some ballot. U_i is closeness-respecting to a set of ballots S if U_i is closeness-respecting to B for all $B \in S$.

This means that when an agent agrees with an outcome on a certain set of issues, when presented with an outcome that also agrees with her on this set of issues (and possibly more), she should not be less satisfied with this new outcome. There are a few interesting things to note here. Firstly, not every utility is closeness-respecting to an element from $\mathsf{Top}(U_i)$. Secondly, if U_i is closeness-respecting to the true preferences B_i of agent i, then it is also top-respecting. Thirdly, if U_i is closeness-respecting to some ballot $\tilde{B}$, U_i is not necessarily closeness-respecting to all $B \in \mathsf{Top}(U_i)$. We will illustrate the latter with an example.

Example 8. Suppose we have two issues and no integrity constraint. Let U_i be given by the average-respecting utility corresponding to the single-ballot utility given by

$$U_i(B) = \begin{cases} 2 & \text{if } b_0 = \mathsf{True} \\ 1 & \text{if } b_0 = \mathsf{False} \text{ and } b_1 = \mathsf{True} \\ 0 & \text{otherwise.} \end{cases}$$

We leave it to the reader to check that U_i is closeness-respecting to $(\mathsf{True}, \mathsf{True})$. Also, $(\mathsf{True}, \mathsf{False}) \in \mathsf{Top}(U_i)$, but we can check that U_i is not closeness-respecting to $(\mathsf{True}, \mathsf{False})$.

We will *not* assume all utility functions are closeness-respecting or average-respecting.

Suppose agent i is an agent with true preferences B_i, who cares about all issues equally much. In this case, her utility function is one that is induced by the Hamming distance, namely

$$U_i^{\mathcal{H}}(B) = m - \mathcal{H}(B_i, B),$$

which gives the number of issues on which B' agrees with her preferences. We call this utility the Hamming utility.

```
uH :: Ballot -> Int -> UtilitySingle
uH tPrefs m b = fromIntegral (m - hamming tPrefs b)
```

It might be that an agent cares more about some issues than about others. To express relative importance of issues to an agent i, we can use a weight function $w_i : \mathcal{I} \to \mathbb{R}_{\geq 0}$.

```
type WeightF = Issue -> Float
```

In Example 1, family member 1 was more inclined on having a cat than a dog. She could have the following weight function.

$$\begin{aligned} w_1(0) &= 1, \\ w_1(1) &= 1, \\ w_1(2) &= 2. \end{aligned}$$

We can encode this.

```
w1 :: WeightF
w1 0 = 1
w1 1 = 1
w1 _ = 2
```

Any weight function induces the *weighted Hamming distance* to a ballot B' and corresponding weighted Hamming utility

$$\begin{aligned} \mathcal{H}^{w_i}(B, B') &= \sum_{j \in \mathcal{I}} |b_j - b'_j| \cdot w_i(j), \\ U_i(B) &= \left(\sum_{j \in \mathcal{I}} w_i(j) \right) - \mathcal{H}^{w_i}(B, B'). \end{aligned}$$

This can be encoded as follows.

```
wHamming :: WeightF -> Int-> Ballot -> Ballot -> Float
wHamming w m b1 b2 = sum (map f [0..(m-1)]) where
  f i = if agree b1 b2 i
        then 0 else w i

uHW :: WeightF -> Ballot -> Int -> UtilitySingle
uHW w tPrefs m b = sum (map w [0..(m-1)]) - wHamming w m tPrefs b
```

In Example 1, the utility for having only a dog is smaller then the utility for having only a cat.

```
  > uHW w1 (i2b 3 7) 3 (i2b 3 3) [True,False,False]
1.0
  > uHW w1 (i2b 3 7) 3 (i2b 3 3) [False,False,True]
2.0
```

Theorem 11. *The (weighted) Hamming utility U_i is closeness-respecting to* $\mathsf{Top}(U_i)$.

Proof. Let U_i be the utility function induced by the weighted Hamming distance $\mathcal{H}^{w_i}$ to $\tilde{B}$. Let $B \in \mathsf{Top}(U_i)$. This means

$$\begin{aligned} U_i(B) &= U_i(\tilde{B}), \\ \sum_{j \in \mathsf{Agree}(\tilde{B},B)} w_i(j) &= \sum_{j \in \mathsf{Agree}(\tilde{B},\tilde{B})} w_i(j), \\ \sum_{j \in \mathsf{Agree}(\tilde{B},B)} w_i(j) &= \sum_{j \in \mathcal{I}} w_i(j), \\ \sum_{j \notin \mathsf{Agree}(\tilde{B},B)} w_i(j) &= 0. \end{aligned}$$

Hence $w_i(j) = 0$ for all $j \notin \mathsf{Agree}(\tilde{B}, B)$. Now we will show that U_i is closeness-respecting to B. Suppose for some rational ballots B', B'' we have

$$\mathsf{Agree}(B, B'') \subseteq \mathsf{Agree}(B, B').$$

By definition of the weighted Hamming utility U_i, we have

$$U_i(B'') = \sum_{j \in \mathsf{Agree}(\tilde{B},B'')} w_i(j).$$

Since $w_i(j) = 0$ for all $j \notin \mathsf{Agree}(\tilde{B}, B)$, we obtain

$$U_i(B'') = \sum_{j \in \mathsf{Agree}(\tilde{B},B'') \cap \mathsf{Agree}(B,B'')} w_i(j).$$

Now since $\mathsf{Agree}(\tilde{B}, B'') \cap \mathsf{Agree}(B, B'') = \mathsf{Agree}(\tilde{B}, B) \cap \mathsf{Agree}(B, B'')$, and since $w_i(j) = 0$ for all $j \notin \mathsf{Agree}(\tilde{B}, B)$, we get

$$U_i(B'') = \sum_{j \in \mathsf{Agree}(B,B'')} w_i(j).$$

Now because $\mathsf{Agree}(B, B'') \subseteq \mathsf{Agree}(B, B')$,

$$U_i(B'') \leq \sum_{j \in \mathsf{Agree}(B,B')} w_i(j).$$

Applying the same things as before we get

$$\begin{aligned} U_i(B'') &\leq \sum_{j \in \mathsf{Agree}(\tilde{B},B')} w_i(j) \\ &= U_i(B'). \end{aligned}$$

Hence all of this gives us $U_i(B'') \leq U_i(B')$, thus U_i is closeness-repecting to $\mathsf{Top}(U_i)$. ☺

Besides these (somehow natural) utilities, there can also be other utilities. For example a utility that formalizes the preference "I would rather have a cat and no dog, but if we do not get a cat I want a dog.".

3.2 Manipulation of the aggregation rule

In the previous chapter we discussed several rules to aggregate agents individual preferences in a group decision. To be able to use these rules, the preferences of the agents need to be known. However, since mind-reading is a skill only possessed by a small group of fortunate ones, the best way to get these preferences is to ask the agents what their preferences are. This causes a vulnerability for the decision making process, since agents can choose to be dishonest about their preferences. If an agent can increase her (expected) utility by lying about her preferences, we say that she can manipulate the aggregation rule. We will first study the case in which the agent has full knowledge about the ballots of the other agents.

3.2.1 Manipulating rules under full knowledge

Using our definition of utility, we obtain the following definition for manipulability of the aggregation rule under full knowledge.

Definition 16 (Manipulability under full knowledge)**.** A rational aggregation rule F is manipulable under full knowledge by agent i with utility U_i at profile $\mathbf{B}$, containing ballot B_i if U_i is top-respecting to B_i and there exists a ballot B_i' such that $U_i(F(\mathbf{B}_{-i}, B_i')) > U_i(F(\mathbf{B}_{-i}, B_i))$.
A rational aggregation function F is manipulable under full knowledge if there exists an agent i, a utility function U_i and a profile $\mathbf{B}$ such that F is manipulable under full knowledge by i at $\mathbf{B}$.

We will call manipulability by (agents with) a closeness-respecting utility, closeness-respecting manipulability.

Some characterization results

Now we do have a definition for manipulability, however this does not give us an easy way to check aggregation rules for manipulability. This is because there is an infinite number of top-respecting utility

functions and we cannot check them all. Therefore we would like to find equivalent requirements for manipulability.

Theorem 12 (Characterization of closeness-respecting manipulability for resolute aggregation rules)**.** *Let F be a* **resolute** *aggregation rule, $\mathbf{B}$ a profile containing ballot B_i.*
There exists an issue $j \in \mathcal{I}$ and a ballot $B_i' \neq B_i$ such that $F(B_i', \mathbf{B}_{-i})_j = \{b_{i,j}\}$, and $F(\mathbf{B})_j \neq \{b_{i,j}\}$ if and only if there exists a utility function U_i that is closeness-respecting to B_i, such that F is manipulable under full knowledge by agent i at profile $\mathbf{B}$.

Proof. Let F be an aggregation rule, $\mathbf{B}$ a profile containing ballot B_i. Suppose there is an issue $j \in \mathcal{I}$ and a ballot $B_i' \neq B_i$ such that $F(B_i', \mathbf{B}_{-i})_j = \{b_{i,j}\}$, and $F(\mathbf{B})_j \neq \{b_{i,j}\}$. Since $B_i' \neq B_i$, there is a j' such that $b'_{i,j'} \neq b_{i,j'}$. If $j = j'$, let

$$w_i(j'') = \begin{cases} 1 & \text{if } j'' = j, \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding Hamming utility is

$$U_i(B) = \begin{cases} 1 & \text{if } b_j = b_{i,j}, \\ 0 & \text{otherwise.} \end{cases}$$

Then since $F(B_i', \mathbf{B}_{-i})_j = \{b_{i,j}\}$ and $F(\mathbf{B})_j \neq \{b_{i,j}\}$, we have $U_i(F(B_i', \mathbf{B}_{-i})) = 1 > 0 = F(\mathbf{B})$. And we have $U_i(B_i) = 1 > 0 = U_i(B_i')$, since $b'_{i,j'} \neq b_{i,j'}$. Hence F is manipulable under full knowledge by agent i with utility U_i, which is a Hamming utility to B_i and thus closeness-respecting to B_i by Theorem 11.
If $j \neq j'$,let

$$w_i(j'') = \begin{cases} 2 & \text{if } j'' = j, \\ 1 & \text{if } j'' = j', \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding Hamming utility is

$$U_i(B) = \begin{cases} 3 & \text{if } b_j = b_{i,j} \text{ and } b_{j'} = b_{i,j'} \\ 2 & \text{if } b_j = b_{i,j} \wedge b_{j'} \neq b_{i,j'}, \\ 1 & \text{if } b_j \neq b_{i,j} \wedge b_{j'} = b_{i,j'}, \\ 0 & \text{otherwise.} \end{cases}$$

Then since $F(B_i', \mathbf{B}_{-i})_j = \{b_{i,j}\}$ and $F(\mathbf{B})_j \neq \{b_{i,j}\}$, we have $U_i(F(B_i', \mathbf{B}_{-i})) \geq 2 > 1 \geq F(\mathbf{B})$. And we have $U_i(B_i) = 3 > 2 \geq U_i(B_i')$, since $b'_{i,j'} \neq b_{i,j'}$. Hence F is manipulable under full knowledge by agent i with utility U_i, which is a Hamming utility to B_i and thus closeness-respecting to B_i by Theorem 11.
For the other direction, suppose there exists a utility function U_i that is closeness-respecting to B_i, such that F is manipulable by agent i at profile $\mathbf{B}$ by agent i. Then there exists a ballot B_i' such that $U_i(F(\mathbf{B}_{-i}, B_i')) > U_i(F(\mathbf{B}_{-i}, B_i))$ and $U_i(B_i') < U_i(B_i)$. Hence since U_i is closeness-respecting, we must have $\mathsf{Agree}(B_i, F(\mathbf{B}_{-i}, B_i')) \subsetneq \mathsf{Agree}(B_i, F(\mathbf{B}_{-i}, B_i))$. Hence there is some j such that $F(B_i', \mathbf{B}_{-i})_j = \{b_{i,j}\}$, and $F(\mathbf{B})_j \neq \{b_{i,j}\}$. ☺

When U_i is not closeness-respecting, the right-to-left direction of the previous theorem does not hold. Consider $m = 2$, $n = 3$, the majority rule Maj and

$$U_0(B) = \begin{cases} 3 & \text{if } B = (1, 0) \\ 2 & \text{if } B = (0, 1) \\ 1 & \text{otherwise.} \end{cases}$$

Now if $\mathbf{B} = ((1,0),(1,1),(0,1))$, $U_0(\mathsf{Maj}(\mathbf{B})) = U_0((1,1)) = 1 < 2 = U_0((0,1)) = U_0(\mathsf{Maj}((0,1),\mathbf{B}_{-0}))$. So agent 0 can manipulate the majority rule under utility U_0. However, the only j for which $\mathsf{Maj}(\mathbf{B})_j \neq \{b_{0,j}\}$ is $j = 1$, however, for no ballot B'_0 we get $\mathsf{Maj}(B'_0, \mathbf{B}_{-0})_1 = b_{0,1}$, since there is already a majority for issue 1 without agent 0. Hence it is not manipulable.

Theorem 13. *If F is a resolute aggregation rule, then F is monotonic and independent if and only if F is not manipulable at any profile containing B_i by an agent with a utility function that is closeness-respecting to ballot B_i.*

Proof. Let F be an aggregation rule that satisfies monotonicity and independence and let agent i have utility U_i that is closeness-respecting to some ballot B_i. Now let $\mathbf{B}$ be any profile containing B_i. Now consider any alternative ballot B'_i and any issue $j \in \mathcal{I}$ such that $F(B'_i, \mathbf{B}_{-i})_j = \{b_{i,j}\}$. Now if $b'_{i,j} = b_{i,j}$, by independence $F(B_i, \mathbf{B}_{-i})_j = F(B'_i, \mathbf{B}_{-i})_j = \{b_{i,j}\}$ and if $b'_{i,j} \neq b_{i,j}$, then by monotonicity, $F(B_i, \mathbf{B}_{-i})_j = \{b_{i,j}\}$. Hence by Theorem 12, F is not manipulable by agent i with utility U_i. Since U_i, $\mathbf{B}$ were arbitrary, F is not manipulable at any profile by an agent with a utility function that is closeness-respecting to some ballot B_i.

For the other direction, suppose first F is not monotonic. Then there is an agent i, issue j, a profile $\mathbf{B}$ containing B_i, and there is an alternative ballot B'_i such that $b_{i,j} = x$, $b'_{i,j} = \neg x$, $F(B_i, \mathbf{B}_{-i})_j = \{\neg x\}$ and $F(B'_i, \mathbf{B}_{-i})_j = \{x\}$. Hence $F(B_i, \mathbf{B}_{-i})_j \neq \{b_{i,j}\}$ and $F(B'_i, \mathbf{B}_{-i})_j = \{b_{i,j}\}$. Thus by Theorem 12, we have that there is a U_i that is closeness-respecting to B_i such that agent i can manipulate F at $\mathbf{B}$.

Now suppose F is not independent. Then by Theorem 2, it is also not weakly independent. Hence there is an issue j, an agent i, a profile $\mathbf{B}$ containing B_i and there is an alternative ballot B'_i such that $b_{i,j} = b'_{i,j}$ and $F(\mathbf{B})_j \neq F(B'_i, \mathbf{B}_{-i})_j$. Now we can assume w.l.o.g. that $F(B'_i, \mathbf{B}_{-i})_j = \{b_{i,j}\} = \{b'_{i,j}\}$ (otherwise we can switch B_i and B'_i). Then we have $F(B_i, \mathbf{B}_{-i})_j \neq \{b_{i,j}\}$, hence by theorem 12, there is a U_i that is top respecting to B_i such that agent i can manipulate F on $\mathbf{B}$.

Hence we have that F is weakly monotonic and weakly independent if and only if F is not manipulable by an agent with a utility function that is closeness-respecting to some ballot B. ☺

This indeed coheres with a result from Dietrich and List (2007b). Note that monotonicity and independence are very strict requirements, and indeed there are few aggregation rules that satisfy both of these. Dietrich and List (2007a) show for resolute rules that if we also require anonymity and responsiveness, these four axioms are satisfied if and only if the rule is a quota rule. The majority rule satisfies both weak monotonicity and (weak) independence hence is not manipulable by agents with a closeness-respecting utility function.

Theorem 14. *For any aggregation rule F, profile $\mathbf{B}$ containing ballot B_i, the following is true: There exists a top-respecting (to B_i) utility U_i such that F is manipulable under full knowledge by agent i, if and only if $F(B_i, \mathbf{B}_{-i}) \neq \{B_i\}$ and there is an alternative ballot B'_i such that $F(B_i, \mathbf{B}_{-i}) \neq F(B'_i, \mathbf{B}_{-i})$.*

Proof. Let F be an aggregation rule, $\mathbf{B}$ a profile containing ballot B_i of agent i. Let U_i be such that agent i can manipulate F at $\mathbf{B}$. Then there is an alternative ballot B'_i such that $U_i(F(B'_i, \mathbf{B}_{-i})) > U_i(F(B_i, \mathbf{B}_{-i}))$ and $U_i(B_i) > U_i(B'_i)$. From the first fact it follows that $F(B_i, \mathbf{B}_{-i}) \neq F(B'_i, \mathbf{B}_{-i})$. From the second combined with the first, since U_i is top respecting and set-consistent, we must have $F(B_i, \mathbf{B}_{-i}) \neq \{B_i\}$.

For the other direction, suppose there is a profile $\mathbf{B}$ containing B_i and there is an alternative ballot B'_i such that $F(B_i, \mathbf{B}_{-i}) \neq F(B'_i, \mathbf{B}_{-i})$ and $F(B_i, \mathbf{B}_{-i}) \neq B_i$. Now suppose $B'_i \neq F(B', \mathbf{B}_{-i})$. Then the

following utility is well defined.

$$U_i(B) = \begin{cases} 2 & \text{if } B = B_i \text{ or } B = F(B_i', \mathbf{B}_{-i}), \\ 1 & \text{if } B = F(B_i, \mathbf{B}_{-i}) \text{ or } B = B_i', \\ 0 & \text{otherwise.} \end{cases}$$

Now we have that U_i is top-respecting to B_i, $U_i(B_i') < U_i(B_i)$ and $U_i(F(B_i', \mathbf{B}_{-i})) > F(B_i, \mathbf{B}_{-i})$, hence F is manipulable under full knowledge by agent i.

In the case that $B_i' = F(B', \mathbf{B}_{-i})$, the following utility is well defined.

$$U_i(B) = \begin{cases} 3 & \text{if } B = B_i \\ 2 & \text{if } B = B_i' \text{ (which equals } F(B', \mathbf{B}_{-i})), \\ 1 & \text{if } B = F(B_i, \mathbf{B}_{-i}) \\ 0 & \text{otherwise.} \end{cases}$$

Now we have again that U_i is top-respecting to B_i, $U_i(B_i') < U_i(B_i)$ and $U_i(F(B_i', \mathbf{B}_{-i})) > F(B_i, \mathbf{B}_{-i})$, hence F is manipulable under full knowledge by agent i. ☺

Corollary 2. *An aggregation rule F is manipulable under full knowledge if and only if there exists a profile $\mathbf{B}$ containing ballot B_i, and an alternative ballot B_i' such that $F(B_i, \mathbf{B}_{-i}) \neq F(B_i', \mathbf{B}_{-i})$ and $F(B_i, \mathbf{B}_{-i}) \neq \{B_i\}$.*

Corollary 3. *If an aggregation rule F is non-manipulable and there are $B_i, B_i', \mathbf{B}$ such that $F(B_i, \mathbf{B}_{-i}) \neq F(B_i', \mathbf{B}_{-i})$, then for all B we must have $F(B, \mathbf{B}_{-i}) = \{B\}$.*

Proof. Suppose there are $B_i, B_i', \mathbf{B}$ such that $F(B_i, \mathbf{B}_{-i}) \neq F(B_i', \mathbf{B}_{-i})$, and there is a B for which $F(B, \mathbf{B}_{-i}) \neq B$. Then we must have that either $F(B, \mathbf{B}_{-i}) \neq F(B_i, \mathbf{B}_{-i})$ or $F(B, \mathbf{B}_{-i}) \neq F(B_i', \mathbf{B}_{-i})$. Hence F is manipulable under full knowledge by Theorem 14. This proves the Corollary. ☺

For the next theorem, we first need a definition.

Definition 17. An aggregation rule F is a *dictatorship* when there is an $i \in \mathcal{N}$ such that $F(\mathbf{B}) = \{B_i\}$ for all rational profiles $\mathbf{B}$.

Theorem 15. *Suppose $|\mathsf{Mod}(\mathrm{IC})| \geq 3$. An aggregation rule F is non-manipulable if and only if it is either constant or a dictatorship.*

Proof. Obviously dictatorships and constant rules are not manipulable. For the other direction, suppose that F is not manipulable and not constant. Since F is not constant, there are profiles $(B_0, \ldots, B_{n-1}), (B_0', \ldots, B_{n-1}')$ such that

$$F(B_0, \ldots, B_{n-1}) \neq F(B_0', \ldots, B_{n-1}').$$

This implies that for some i, we must have

$$F(B_0', \ldots, B_{i-1}', B_i, \ldots, B_{n-1}) \neq F(B_0', \ldots, B_i', B_{i+1}, \ldots, B_{n-1}).$$

Now let $\mathbf{B}''_{-i} = B_0', \ldots, B_{i-1}', B_{i+1}, \ldots, B_{n-1}$. Since F is not manipulable and $F(B_i, \mathbf{B}''_{-i}) \neq F(B_i', \mathbf{B}''_{-i})$, by Corollary 3, this implies that $F(B, \mathbf{B}''_{-i}) = \{B\}$ for all $B \in \mathsf{Mod}(\mathrm{IC})$. Now consider some arbitrary profile $\mathbf{B}^*_{-i} = B_0^*, \ldots, B_{n-1}^*$. We define the following function

$$G(B, k)_l = \begin{cases} B & \text{if } l = i, \\ B_l^* & \text{if } l \neq i \text{ and } l \leq k, \\ B_l'' & \text{if } l \neq i \text{ and } l > k. \end{cases}$$

We have that $G(B,k)$ defines a profile. We will prove that for any $k \in \mathcal{I}$, $F(G(B,k)) = \{B\}$ for all $B \in \mathsf{Mod}(\mathrm{IC})$, by the induction on k.

For $k = 0$, $G(B,0) = B, \mathbf{B}''_{-i}$, hence we have $F(G(B,0)) = \{B\}$.

suppose $F(G(B,k)) = \{B\}$ for all $B \in \mathsf{Mod}(\mathrm{IC})$. Now if $k+1 = i$, we have $G(B,k) = G(B,k+1)$, hence $F(G(B,k)) = F(G(B,k+1))$.

If $k+1 \neq i$, let $A_1 \neq A_2 \in \mathsf{Mod}(\mathrm{IC})$ be unequal to B''_{k+1} (these exist since $|\mathsf{Mod}(\mathrm{IC})| \geq 3$). Then by induction hypothesis, we have

$$F(G(A_1,k)) = \{A_1\},$$
$$F(G(A_2,k)) = \{A_2\}.$$

By definition of G we have $G(A_1,k)_{k+1} = B''_{k+1} = G(A_2,k)_{k+1}$.

Now Since $F(G(A_1,k)) = \{A_1\} \neq \{G(A_1,k)_{k+1}\}$ and $G(A_1,k+1) = B^*_{k+1}, G(A_1,k)_{-(k+1)}$, by Corollary 3 contraposed, we must have

$$F(G(A_1,k+1)) = F(G(A_1,k)) = \{A_1\},$$

and similarly

$$F(G(A_2,k+1)) = F(G(A_2,k)) = \{A_2\}.$$

Thus

$$F(G(A_1,k+1)) \neq F(G(A_2,k+1))$$

Hence by Corollary 3, for any B we must have $F(G(B,k+1)) = F(B, G(A_1,k+1)_{-i}) = B$.
This shows by induction that $F(G(B,k)) = B$ for all $B \in \mathsf{Mod}(\mathrm{IC})$ and for all $k \in \mathcal{I}$. We can conclude from this that also $F(B, \mathbf{B}^*_{-i}) = F(G(B,n-1)) = B$ for all $B \in \mathsf{Mod}(\mathrm{IC})$, and since $\mathbf{B}^*$ was arbitrary, agent i is a dictator. ☺

At first sight, the result of Theorem 15 looks similar to the Gibbard-Satterthwaite Theorem. However mathematically the two results are very different. Here the impossibility is more direct because of the extreme freedom in picking a utility function that works.

3.2.2 Manipulating rules under partial knowledge

Given that the utility function expresses the agents satisfaction with a certain outcome, any agent would like to have an outcome such that the utility is as high as possible. As we showed above, all reasonable rules are susceptible to manipulation. These manipulations are quite straightforward, however, they all require full knowledge about the preferences of the other agents. In real life this will almost never be the case. We will now investigate manipulations by agents that do not have full knowledge.[2]

In this thesis we will only consider identically distributed beliefs. That is, an agent has a belief about what the ballots of the other agents could be, and her ignorance about the remaining possibilities is given by a uniform distribution. For example, she might know that her neighbour will vote True

[2]We discuss the case in which agents have partial knowledge on the ballots of other agents. One could also consider cases in which agents have knowledge on the true preferences of other agents, including their utility functions. This would give rise to a bayesian game which would be an interesting line of research.

for issue 1, therefore she knows that all profiles in which her neighbour votes False for issue 1 are not possible. We could easily extend this approach by adding probabilities to our knowledge. The reason that we will not do this here is twofold. First of all, since the number of possible profiles grows exponentially with the number of agents and issues, assigning separate probabilities to each of these will be a complicated job for an agent. Secondly, this would make the illustrative examples in this thesis harder to comprehend. Thus, in this thesis, the knowledge $k_i \in \mathcal{P}((\mathsf{Mod}(\mathrm{IC})^i \times \mathsf{Mod}(\mathrm{IC}))^{n-i-1})$ of an agent i consists of a list of possible $\min_i$-profiles. Note that this is just knowledge about the ballots agents cast and does not take into account meta-knowledge such as knowledge about the utilities or the knowledge of other agents.

```
type Knowledge = [MinIProfile]
```

We encode a function to split multiple profiles into $\min_i$-profiles.

```
splitProfiles :: Agent -> [Profile] -> Knowledge
splitProfiles i = map (splitProfile i)
```

We can use this function to create knowledge sets. Consider Example 6. If agent 2 is fully ignorant about the preferences of the other agents (except that they are rational), we can encode this as follows.

```
exampleKn1 :: Knowledge
exampleKn1 = nub (splitProfiles 2 (allProfiles 5 3 l0))
```

It will turn out that we can construct cases where manipulation under full ignorance is possible. Thus this knowledge set is an interesting one.

Suppose on the other hand that agent 2 knows her family members will vote True for issues $2, 3$ and 4, but she does not know their opinions on issues 0 and 1 (note that their preferences on these are not forced completely by their votes on issues 2, 3 and 4). In this case her knowledge is given by `exampleKn2`.

```
exampleKn2 :: Knowledge
exampleKn2 = let
    f x = and (drop 2  (x!!1) ++
              drop 2 (head x))
  in
    nub (splitProfiles 2 (filter f (allProfiles 5 3 l0)))
```

When having a meeting, announcements can be made that trigger a knowledge update. For example, if agent 1 says: "I am going to vote True for issue 1" and I believe her, I want to update my knowledge, by deleting all profiles in which she votes False for issue 1.

```
updateKnowledge :: Int -> Agent -> Issue ->
                Vote -> Knowledge -> Knowledge
updateKnowledge m a i vote kn = let
    j = length (fst (head kn))
    n = length (uncurry (++) (head kn)) + 1
  in
    if null (checkIndices (m,n,j,a,i))  then let
        update (x,y) = ((x ++ [i2b m 0] ++ y) !! a) !! i == vote
      in
        filter update kn
    else
  error (unlines (checkIndices (m,n,j,a,i)))
```

Not all possible knowledge updates can be done with this function, for example after the statement "I will vote True for issue 0 if I vote true for issue 1", we would need a more complicated knowledge update function. The reader is encouraged to implement such a function. For our examples the knowledge update function that was mentioned above will do.

Example 6 - Spending money (continued). If family member 0 announces that she will vote True for issue 0, family member 2 wants to update her knowledge. After updating, the set of possible profiles is smaller.

```
  > length exampleKn2
9
  > length (updateKnowledge 5 0 0 True exampleKn2)
3
```

■

Using knowledge sets, we can calculate the expected utility E_F of the outcome of F when casting ballot B, given knowledge k. This is given by

$$E_F(U_i, B, k_i) = \frac{\sum_{(\vec{a},\vec{b})\in k_i} U_i(F(\vec{a}, B, \vec{b})}{|k_i|}$$

We encode this.

```
expectedUBinA :: Ballot -> UtilitySet -> Knowledge
              -> Int -> IC -> AR -> Float
expectedUBinA b u kn m bc rule = let
    know = map (\(x,y) -> (x ++ [b] ++ y)) kn
    utility x = u  (rule m bc x)
  in
    sum (map utility know) / fromIntegral (length know)
```

Example 6 - Spending money (continued). The preferences of agent 2 are represented by the integer Ballot `8`. Suppose all questions are equally important to her, so her utility function is `uH (i2b 5 8) 5`. She is fully ignorant, which is represented by the knowledge `exampleKn1`. We can check the expected utilities for reporting her (honest or dishonest) preferences in the priority rule. For example, we can compare her expected utilities for her honest ballot 8 and her dishonest ballot 24.

```
  > expectedUBinA (i2b 5 24) (uSet (uH (i2b 5 8) 5)) exampleKn1 5 l0 priority
3.4624
  > expectedUBinA (i2b 5 8) (uSet (uH (i2b 5 8) 5)) exampleKn1 5 l0 priority
3.7696
```

Clearly under full ignorance, reporting her true preferences is better than reporting the false preferences represented by the integer 24. However, if we consider the case in which she knows that the other family members will vote True for issues 2,3 and 4, we get the following expected utilities.

```
  > expectedUBinA (i2b 5 24) (uSet (uH (i2b 5 8) 5)) exampleKn2 5 l0 priority
1.6666666
  > expectedUBinA (i2b 5 8) (uSet (uH (i2b 5 8) 5)) exampleKn2 5 l0 priority
1.4444444
```

We see that even under full ignorance, the expected utility for family member 2 is higher if she misreports her true preferences by casting ballot 24 instead of ballot 8. ■

Definition 18 (Manipulability under partial knowledge)**.** An aggregation rule F, is called manipulable by agent i under knowledge k_i, if there is a ballot B_i', such that $E_F(U_i, B_i', k_i) > E_F(U_i, B_i, k_i) \quad \forall B_i \in \mathsf{Top}(U_i)$.

As shown above, sometimes we can also manipulate under partial knowledge. The minimal amount of knowledge we can have is when we are fully ignorant about the preferences of the others (in which case our knowledge set is maximal).

Definition 19. The *full-ignorance set* of agent i for the integrity constraint IC is the set $\mathsf{Mod}(\mathrm{IC})^i \times \mathsf{Mod}(\mathrm{IC})^{n-i-1}$ denoting all possible $\min_i$-profiles $\mathbf{B}_{-i}$ given the integrity constraint IC. An agent is said to be *fully ignorant* if her knowledge is the full-ignorance set. A knowledge set k_i is called the *full-knowledge set* if k_i has only one element.

Note that a knowledge set should not be empty, since this can only be the case when the agents knowledge is inconsistent.

For any IC, m, n we can create the full-ignorance set.

```
noKnowledge :: Int -> Int -> Agent -> IC -> Knowledge
noKnowledge m n a bc = nub (splitProfiles a (allProfiles m n bc))
```

Definition 20. An aggregation rule F is called full-ignorance manipulable if there exists a set of agents $\mathcal{N}$, an agent $i \in \mathcal{N}$, some utility U_i of agent i, such that F is manipulable by i under the full-ignorance set.

When we have full knowledge, our definitions for manipulability under full and partial knowledge coincide.

Theorem 16. *When k_i is a full-knowledge set $\{\mathbf{B}_{-i}\}$, F is manipulable by agent i under knowledge k_i if and only if F is manipulable under full knowledge by agent i at the profile $B_i, \mathbf{B}_{-i}$, for all $B_i \in \mathsf{Top}(U_i)$.*

Proof. Suppose F is manipulable by agent i under knowledge $k_i = \{\mathbf{B}_{-i}\}$. Then there is a ballot B_i', such that $E_F(U_i, B_i', k_i) > E_F(U_i, B_i, k_i) \quad \forall B_i \in \mathsf{Top}(U_i)$. Hence

$$U_i\left(\frac{F(B_i', \mathbf{B}_{-i})}{|k_i|}\right) > U_i\left(\frac{F(B_i, \mathbf{B}_{-i})}{|k_i|}\right) \quad \forall B_i \in \mathsf{Top}(U_i)$$
$$U_i(F(B_i', \mathbf{B}_{-i})) > U_i(F(B_i, \mathbf{B}_{-i})) \quad \forall B_i \in \mathsf{Top}(U_i).$$

So we must have $F(B_i', \mathbf{B}_{-i}) \neq F(B_i, \mathbf{B}_{-i}) \quad \forall B_i \in \mathsf{Top}(U_i)$. Hence F is manipulable under full knowledge by agent i at the profile $B_i, \mathbf{B}_{-i}$ for all $B_i \in \mathsf{Top}(U_i)$. The other direction is proven similarly. ☺

We can use the expected utility to calculate which ballot from a given set of ballots maximizes the expected utility.

```
bestBallots :: [Ballot] -> UtilitySet -> Knowledge
               -> Int -> IC -> AR -> [Ballot]
bestBallots ballots ui kn m bc rule
  | null kn = error "\nKnowledge is empty.\n"
  | otherwise = maximaBy (comparing f) ballots where
      f b = expectedUBinA  b ui kn m bc rule
```

In Example 6, it is beneficial for agent 2 to misreport her preferences.

```
  > bestBallots (models 5 10) (uSet (uH (i2b 5 8) 5)) exampleKn2 5 10
  priority
[[True,True,False,False,False]]
```

When the family would have used the distance based rule, it would not have been beneficial for agent 2 to misreport her preferences, however lots of dishonest ballots would have given her the same expected utility.

```
  > bestBallots (models 5 10) (uSet (uH (i2b 5 8) 5)) exampleKn2 5
  10 dbRule
[[False,True,True,True,True],
[False,True,True,True,False],
[False,True,True,False,True],
[False,True,True,False,False],
[False,True,False,True,True],
[False,True,False,True,False],
[False,True,False,False,True],
[False,True,False,False,False]]
```

3.2.3 A manipulation tool

To help the manipulating agent, we will combine everything from the previous sections into an interactive environment. First we need a function to update the set of ballots the agent can choose from. For

example, it might be that she already voted on the first issue, hence she cannot choose from all ballots $B \in \mathsf{Mod}(\mathrm{IC})$ anymore.

```
updateBallotSet :: [Ballot] -> Issue -> Vote -> [Ballot]
updateBallotSet ballots i vote =
  filter (\x -> x !! i == vote) ballots
```

We write some functions to get user input.

```
getAgent, getIss, getVote :: IO String
getAgent = get "\nAgent :\n"
getIss = get "\nIssue :\n"
getVote = get "\nVote :\n"

typeErr :: String
typeErr = "\nIncorrect type for input, try again\n"
```

Now given m, n, agent i, her utility U_i, the integrity constraint IC and the aggregation rule F, we can manipulate the aggregation rule under partial knowledge.

```
manipulateAR :: Int -> Int -> Agent
                -> UtilitySet -> IC -> AR -> IO()
manipulateAR m n j uj bc rule =
  manipulate startBallots startKnowledge where
    startKnowledge = nub (splitProfiles j (allProfiles m n bc))
    startBallots = models m bc
    manipulate ballots kn = do
       putStr ("\n Press u to update knowledge,"
                ++ " r to restrict own ballot,"
                ++ " b to calculate best ballot,"
                ++ " q to quit, k to show knowledge,"
                ++ " s to show set of possible ballots\n")
       l <- getLine
       case l of
         "b" -> print (bestBallots ballots uj kn m bc rule)
         "q" -> putStr "\nBye\n"
         "k" -> do
           print kn
           manipulate ballots kn
         "s" -> do
           print ballots
           manipulate ballots kn
         "r" -> do
           i <- getIss
           vote <- getVote
           case (readMaybe i :: Maybe Issue,
                 readMaybe vote :: Maybe Vote) of
              (Just i', Just vote') -> do
```

```
                let newBallots = updateBallotSet ballots i' vote'
                manipulate newBallots kn
              _ -> do
                putStr typeErr
                manipulate ballots kn
        _ -> do
          a <- getAgent
          i <- getIss
          vote <- getVote
          case (readMaybe a :: Maybe Agent,
                readMaybe i ::  Maybe Issue,
                readMaybe vote :: Maybe Vote) of
             (Just a', Just i', Just vote') ->
               if null (checkIndices (m,n,j,a',i')) then do
                 putStr (unlines (checkIndices (m,n,j,a',i')) ++
                         "Try again\n")
                 manipulate ballots kn
                 else do
                 let newKnowledge =
                       updateKnowledge m a' i' vote' kn
                 manipulate ballots newKnowledge
             _ -> do
                    putStr typeErr
                    manipulate ballots kn
```

Example 6 - Spending money (continued). We can apply this interactive implementation on Example 6. We start with full ignorance, thus all rational ballots are still possible and equally likely. We find out agent 0 votes True for issue 0, hence we update our knowledge. After that we calculate the optimal ballot to cast.

```
  > manipulateAR 5 3 2 (uSet (uH (i2b 5 8) 5)) l0 priority

 Press u to update knowledge, r to restrict own ballot,
 b to calculate best ballot, q to quit, k to show knowledge,
 s to show set of possible ballots
?> u

Agent :
?> 0

Issue :
?> 0

Vote :
?> True

 Press u to update knowledge, r to restrict own ballot,
 b to calculate best ballot, q to quit, k to show knowledge,
 s to show set of possible ballots
?> b
[[False,True,False,False,False]]
```

■

This is of course just a short example of how this works. The reader should feel free to try and do some manipulations with this herself.

3.2.4 Manipulating social choice functions

In social choice functions for preference aggregation, manipulation works in a similar way. Here we will introduce the necessary implementations to be able to deal with these manipulations. We also provide some definitions. Definitions for manipulability under full/partial knowledge, expected utility, etcetera transfer in a straightforward way from the definitions for binary aggregation. We will not spell out these formal definitions, but refer to the implementation instead.

Definition 21. A utility function on alternatives, $U_i : \mathcal{P}(\chi) \to \mathbb{R}$ for an agent $i \in \mathcal{N}$, is a function, assigning to each set of alternatives S a number, expressing the agents satisfaction with S as an outcome.

As before, we write $U_i(a)$ for $U_i(\{a\})$.

```
type UtilitySCSingle = Alternative -> Float
type UtilitySCSet = [Alternative] -> Float

uSet2 :: UtilitySCSingle -> UtilitySCSet
uSet2 u alts = sum (map u alts) / fromIntegral (length alts)
```

A utility function on alternatives u_i is assumed to be *order-respecting* to the true preference ordering of agent i. That is, if an agent (honestly) prefers a over b or likes them both equally, then we must have $U_i(a) \geq U_i(b)$. Therefore the set of true preference rankings of agent i should satisfy

$$\mathsf{Top}(U_i) = \{B \in \mathsf{ModIC} \mid B_{(a,b)} = \mathsf{True} \leftrightarrow U_i(a) \geq U_i(b) \quad \forall a, b \in \chi\}.$$

```
truePrefsSC :: Int -> UtilitySCSingle -> [Preference]
truePrefsSC v ui = let
    combs = [(a,c)| a <- [0..(v-1)], c <- [0..(v-1)]]
    test b (a,c) = (ui a == ui c) ||
                   (b !! alt2iss (a,c) v == (ui a > ui c))
    f b = all (test b) combs
  in
    profile2prefs (v*v)
    (bs2is (filter f (models (v*v) (lo0 v))))
```

Example 5 - Painting a house (continued). Consider Example 5 about the family that had to decide what color to paint their house. Agent 0 could have the following utility function on single alternatives.

```
u0 :: UtilitySCSingle
u0 0 = 4
```

```
u0 1 = 1
u0 _ = 0
```

If we assume the utility is average-respecting, we can extend this naturally to a utility function on sets. ■

As with aggregation functions, we can calculate the expected utility given a certain knowledge, ballot, utility and social choice function.

```
expectedUSCF :: Ballot -> UtilitySCSet -> Knowledge
              -> Int -> SCF -> Float
expectedUSCF b u kn v rule = let
    know = map (\(x,y) -> (x ++ [b] ++ y)) kn
    utility x =  u (rule v x)
  in
    sum (map utility know) / fromIntegral (length know)
```

Now suppose agent 0 is fully ignorant. We can encode this as follows.

```
exampleKn3 :: Knowledge
exampleKn3 = nub (splitProfiles 0 (allProfiles 9 3 (lo0 3)))
```

We can calculate her expected utility.

```
  > expectedUPrA 200 (uSet2 u0) exampleKn3 3 copelandRule
2.8888888
```

Of course this fact alone is not very interesting, so we will use this again to find what ballot agent 0 should cast to get the best expected utility.

```
bestPrefs :: [Ballot] -> UtilitySCSet -> Knowledge
              -> Int -> SCF -> [Preference]
bestPrefs ballots u kn v rule = let
    f b = expectedUSCF  b u kn v rule
    sortedBallots = sortBy (comparing (\x -> u [b2i x])) ballots
  in
    ballots2prefs v (maximaBy (comparing f) sortedBallots)
```

We use this to calculate what ballot agent 0 should cast.

```
  > bestPrefs (models 9 (lo0 3)) (uSet2 u0) exampleKn3 3 copelandRule
[[0,1,2]]
```

So with this knowledge, it is clearly best for agent 0 to cast a ballot representing her true preferences. This is not surprising as the Copeland rule satisfies the axiom of neutrality with respect to alternatives, and in the integrity constraint, alternatives are also interchangable, thus under full ignorance, no difference can really be made between alternatives. Let us now see what happens if agent 0 has some information, namely she knows the ballot of agent 1.

```
exampleKn4 :: Knowledge
exampleKn4 = splitProfiles 0 (filter (\x -> x !! 1 == i2b 9 104)
          (allProfiles 9 3 (lo0 3)))
```

We determine the best preference ordering for agent 0 and we calculate the corresponding expected utility.

```
  > bestPrefs (models 9 (lo0 3)) (uSet2 u0) exampleKn4 3 copelandRule
[[0,2,1]]
  > pref2int [0,2,1]
194
  > expectedUSCF (i2b 9 194) (uSet2 u0) exampleKn4 3 copelandRule
2.6111112
```

We can calculate the expected utility for casting an honest ballot.

```
   > expectedUSCF (i2b 9 200) (uSet2 u0) exampleKn4 3 copelandRule
  2.5
```

Even without having any knowledge about the preferences of agent 2, agent 0 should cast the insincere ballot `[0,2,1]`, which has a higher expected utility than her sincere ballot `[0,1,2]`.

When updating knowledge in case of preference aggregation (for either social choice, or social welfare functions), we would like to express our knowledge in terms of the pairs (a, b).

```
updateKnowledgePA :: Int -> Agent ->
                     (Alternative, Alternative) -> Vote
                     -> Knowledge -> Knowledge
updateKnowledgePA v a (i,j) =
  updateKnowledge (v * v) a (alt2iss (i,j) v)
```

Similarly as for binary judgement aggregation rules, we can put everything together to obtain a tool for a manipulating agent. First we introduce a function to get a pair of alternatives from the user.

```
getAlts :: IO (String,String)
getAlts = do a <- get "\nPrefers a over b. What is a?\n"
             b <- get "\n What is b? \n"
             return (a,b)
```

We check whether the alternatives are indeed smaller than the total number of alternatives.

```
checkIndicesAlts :: (Alternative,Alternative,Int) -> String
checkIndicesAlts (a,b,v)
  | a > v-1 || b > v-1 =
    "Alternative index too large\n"
  | a < 0 || b < 0 =
    "Alternative index too small\n"
  | otherwise = ""
```

We encode a manipulation tool for social choice functions.

```
manipulateSCF :: Int -> Int -> Agent
                 -> UtilitySCSet -> SCF -> IO()
manipulateSCF v n j uj rule =
  manipulate startBallots startKnowledge where
    m = v * v
    startKnowledge = nub (splitProfiles j (allProfiles m n (lo0 v)))
    startBallots = models m (lo0 v)
    manipulate ballots kn = do
       putStr ("\n Press u to update knowledge,"
                ++ " r to restrict own ballot,"
                ++ " b to calculate best preference order,"
                ++ " q to quit, k to show knowledge,"
                ++ " s to show set of possible ballots\n")
       l <- getLine
       case l of
         "b" -> print (bestPrefs ballots uj kn v rule)
         "q" -> putStr "\nBye\n"
         "k" -> do
           print kn
           manipulate ballots kn
         "s" -> do
           print ballots
           manipulate ballots kn
         "r" -> do
           (p1,p2) <- getAlts
           case (readMaybe p1 :: Maybe Alternative,
                 readMaybe p2 :: Maybe Alternative) of
              (Just p1', Just p2') -> do
                let i = alt2iss (p1',p2') v
                let newBallots = updateBallotSet ballots i True
                manipulate newBallots kn
              _ -> do
                putStr typeErr
                manipulate ballots kn
         _ -> do
           a <- getAgent
           (p1,p2) <- getAlts
           case (readMaybe a :: Maybe Agent,
                 readMaybe p1 :: Maybe Alternative,
```

```
                readMaybe p2 :: Maybe Alternative) of
            (Just a', Just p1', Just p2') -> do
              let i = alt2iss (p1',p2') v
              if null (checkIndices (m,n,j,a',i)) &&
                null (checkIndicesAlts (p1',p2',v)) then do
                let newKnowledge =
                      updateKnowledge m a' i True kn
                manipulate ballots newKnowledge
                else do
                  putStr (unlines (checkIndices (m,n,j,a',i)) ++
                          checkIndicesAlts (p1',p2',v) ++
                          "Try again\n")
                  manipulate ballots kn
            _ -> do
                   putStr typeErr
                   manipulate ballots kn
```

We can apply this to our example. First agent 2 finds out agent 0 prefers yellow over blue. She updates her knowledge. Then agent 2 states that she prefers red over yellow. To stay true to herself she restricts her own ballot. After this they decide to vote. We can calculate her best preference.

```
  > manipulateSCF 3 3 2 (uSet2 u0) copelandRule

 Press u to update knowledge, r to restrict own ballot,
 b to calculate best preference order, q to quit,
 k to show knowledge, s to show set of possible ballots
?> u

Agent :
?> 0

Prefers a over b. What is a?
?> 2

 What is b?
?> 0

 Press u to update knowledge, r to restrict own ballot,
 b to calculate best preference order, q to quit,
 k to show knowledge, s to show set of possible ballots
?> r

Prefers a over b. What is a?
?> 1

 What is b?
?> 2

 Press u to update knowledge, r to restrict own ballot,
 b to calculate best preference order, q to quit,
 k to show knowledge, s to show set of possible ballots
```

```
?> b
[[0,1,2]]
```

The interactive implementation above can be used to do any manipulation on social choice functions. We encourage the reader to try it.

3.2.5 How much knowledge is needed for manipulation of the aggregation rule?

Now that we know how we can manipulate aggregation rules with partial knowledge, we can investigate how much knowledge is actually needed for manipulation. It turns out that in a lot of cases, that is none at all.

First of all, note that having more knowledge does not necessarily mean that your manipulation will be more effective.

Example 1 - Cat or Dog (continued). Suppose agent 2 has the following utility function.

```
uDC2 :: UtilitySet
uDC2 = uSet (uDC2'.b2i) where
  uDC2' 1 = 100
  uDC2' 7 = 80
  uDC2' 4 = 70
  uDC2' _ = 0
```

Note that this function is top-respecting to the honest ballot of agent 2, represented by the integer 1. under full ignorance, we calculate the best ballot for agent 2.

```
  > bestBallots (models 3 q0) uDC2 (noKnowledge 3 3 2 q0) 3 q0 priority
[[True,True,True]]
```

This ballot is represented by the integer 1. What would the outcome be if she would cast this ballot?

```
  > priority 3 q0 (is2bs 3 [4,7,7])
[[True,True,True]]
```

We calculate her utility for this outcome.

```
  > uDC2 [[True,True,True]]
80.0
```

Now suppose agent 2 knows the ballot of agent 1, her best ballot changes.

```
  > bestBallots (models 3 q0) uDC2 (filter (\(_:y:_,_) -> y == i2b 3 7)
  (noKnowledge 3 3 2 q0)) 3 q0 priority
[[False,False,True]]
```

This ballot is her true ballot in the profile `exampleA`. However, if she would cast this ballot, she would end up with a lower utility.

```
  > uDC2 (priority3 q0 exampleA)
70.0
```

Thus, having knowledge about the ballot of agent 0 did not help agent 2 here. ■

The fact that having more knowledge is not always beneficial seems counter-intuitive. Gaining knowledge about the other ballots should help for manipulation. Of course having full knowledge is optimal; then you can ensure a maximal utility. However, other knowledge might be misleading (e.g. you accidentally obtain knowledge that is not representative for the total profile). If we do not get misleading knowledge on purpose, we can expect our calculations of the expected utility to become on average more accurate when we get more knowledge, since the average of a set of values should on average be closer to the individual elements of the set when the set is smaller.

Example 6 - Spending money (continued). Suppose family member 2 is fully ignorant about the preferences of the other agents. Also assume that issues 2,3 and 4 are much more important to her than issues 0 and 1. This can be represented by the following weight function on issues.

```
w2 :: WeightF
w2 0 = 1
w2 1 = 1
w2 2 = 100
w2 3 = 100
w2 _ = 100

uExHome :: UtilitySet
uExHome = uSet (uHW w2 (i2b 5 8) 5)
```

Now if we use the distance based rule, we see that even under full ignorance reporting her false preferences represented by integer 24, gives a higher expected utility than reporting her true preferences represented by 8.

```
  > expectedUBinA (i2b 5 8) uExHome (noKnowledge 5 3 2 10) 5 10 dbRule
232.3408
  > expectedUBinA (i2b 5 24) uExHome (noKnowledge 5 3 2 10) 5 10 dbRule
234.7568
```
■

A more striking example is that even the majority rule, that we somehow see as a perfect rule, can be manipulable under full ignorance.

Example 9. Suppose $n = 3$ family members vote on the issues "cat" and "dog" (but this time we do not have an issue "cat and dog"). We see that there is no integrity constraint, thus we can apply the majority rule. We consider the utility function `u0ExMaj` of agent 0 on the ballots.

```
u0ExMaj :: UtilitySet
u0ExMaj = uSet (u0ExMaj'.b2i) where
  u0ExMaj' 2 = 100
  u0ExMaj' 3 = 90
  u0ExMaj' 1 = 55
  u0ExMaj' _ = 1
```

Thus, agent 0 wants a cat and a dog, but if that is not possible, she'd rather have a cat and no dog than a dog and no cat or no pets at all. Clearly the true preferences of agent 0 are given by the ballot $(\mathsf{True}, \mathsf{False})$, represented by 3.

```
  > bestBallots (models 2 z0) u0ExMaj (noKnowledge 2 3 0 n0) 2 n0 majResolveAll
[[True,True]]
```

With this utility function, her best pick for a ballot is one that does not represent her true preferences. Hence even when the majority rule is collectively rational for some integrity constraint, we can not be sure manipulations will not happen. ■

We will now investigate which rules are and which are not manipulable under full ignorance. The following theorem gives a characterization for this.

Theorem 17. *Let F be an aggregation rule, B_i be a ballot for agent i. Then there exists a utility U_i top-respecting to B_i such that F is full-ignorance manipulable by agent i if and only if there is an alternative ballot $B_i' \neq B_i$ and some set of ballots $S \neq \{B_i\}$ such that $|\{\mathbf{B}_{-i} \mid F(B_i', \mathbf{B}_{-i}) = S\}| < |\{\mathbf{B}_{-i} \mid F(B_i, \mathbf{B}_{-i}) = S\}|$.*

Proof. Let k_0 denote the full-ignorance set. Suppose for all $B_i' \neq B_i$ and every $S \subseteq \mathsf{Mod}(\mathrm{IC})$ such that $S \neq \{B_i\}$ we have $|\{\mathbf{B}_{-i} \mid F(B_i', \mathbf{B}_{-i}) = S\}| \geq |\{\mathbf{B} \mid F(B_i, \mathbf{B}_{-i}) = S\}|$.

Now first note that

$$\sum_{S \subseteq \mathsf{Mod}(\mathrm{IC})} |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}'_{-i}) = S\}| =$$
$$\sum_{S \subseteq \mathsf{Mod}(\mathrm{IC})} |\{\mathbf{B}'_{-i} \mid F(B_i', \mathbf{B}'_{-i}) = S\}| = |\mathsf{Mod}(\mathrm{IC})^{n-1}|,$$

Hence we must have

$$\sum_{S \in \mathcal{P}(\mathsf{Mod}(\mathrm{IC})) \setminus \{\{B_i\}\}} |\{\mathbf{B}'_{-i} \mid F(B_i', \mathbf{B}'_{-i}) = S\}| - \sum_{S \in \mathcal{P}(\mathsf{Mod}(\mathrm{IC})) \setminus \{\{B_i\}\}} |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}'_{-i}) = S\}|$$
$$= |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}_{-i}) = \{B_i\}\}| - |\{\mathbf{B}'_{-i} \mid F(B_i', \mathbf{B}_{-i}) = \{B_i\}\}|$$

Thus we also have

$$\sum_{S \in \mathcal{P}(\mathsf{Mod}(\mathrm{IC})) \setminus \{\{B_i\}\}} |\{\mathbf{B}'_{-i} \mid F(B_i', \mathbf{B}'_{-i}) = S\}| - |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}'_{-i}) = S\}|$$
$$= |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}_{-i}) = \{B_i\}\}| - |\{\mathbf{B}'_{-i} \mid F(B_i', \mathbf{B}_{-i}) = \{B_i\}\}|.$$

And since for all $B_i' \neq B_i$ and every $S \subseteq \mathsf{Mod}(\mathrm{IC})$ such that $S \neq \{B_i\}$ we have $|\{\mathbf{B}_{-i} \mid F(B_i', \mathbf{B}_{-i}) = S\}| \geq |\{\mathbf{B} \mid F(B_i, \mathbf{B}_{-i}) = S\}|$, we must have $|\{\mathbf{B}'_{-i} \mid F(B_i', \mathbf{B}'_{-i}) = S\}| - |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}'_{-i}) = S\}| \geq 0$

and thus also $|\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}_{-i}) = \{B_i\}\}| - |\{\mathbf{B}'_{-i} \mid F(B'_i, \mathbf{B}_{-i}) = \{B_i\}\}| \geq 0$. Hence since for any U_i that is top respecting to B_i, we have $U_i(B_i) \geq U_i(S)$ for any $S \subseteq \mathsf{Mod}(\mathrm{IC})$. Hence we have

$$\begin{aligned}\sum_{S \in \mathcal{P}(\mathsf{Mod}(\mathrm{IC})) \setminus \{\{B_i\}\}} & U_i(S) \cdot (|\{\mathbf{B}'_{-i} \mid F(B'_i, \mathbf{B}'_{-i}) = S\}| - |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}'_{-i}) = S\}|) \\ & \leq U_i(B_i) \cdot (|\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}_{-i}) = \{B_i\}\}| - |\{\mathbf{B}'_{-i} \mid F(B'_i, \mathbf{B}_{-i}) = \{B_i\}\}|).\end{aligned}$$

This gives us

$$\sum_{S \subseteq \mathsf{Mod}(\mathrm{IC})} U_i(B) \cdot |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}'_{-i}) = S\}| \geq \sum_{S \subseteq \mathsf{Mod}(\mathrm{IC})} U_i(B) \cdot |\{\mathbf{B}'_{-i} \mid F(B'_i, \mathbf{B}'_{-i}) = S\}|,$$

hence

$$E_F(U_i, B_i, k_0) \geq E_F(U_i, B'_i, k_0),$$

Thus F is not full-ignorance manipulable by agent i with top respecting utility U_i.

For the other direction, suppose there is a pair of alternative ballots $B_i \neq B'_i$ and some $S \neq \{B_i\}$ such that, $|\{\mathbf{B}'_{-i} \mid F(B'_i, \mathbf{B}'_{-i}) = S\}| < |\{\mathbf{B}'_{-i} \mid F(B_i, \mathbf{B}'_{-i}) = S\}|$.

Let l be the number of possible $\min_i$-profiles (hence $l = |k_0|$). We let the utility U_i of agent i be

$$U_i(S') = \begin{cases} 2l & \text{if } S' = \{B_i\}, \\ = 0 & \text{if } S' = S, \\ = 2l - 1 & \text{otherwise.} \end{cases}$$

Since $S \neq \{B_i\}$, this is well-defined. Also we have $U_i(B_i) > U_i(B'_i)$ and

$$E_F(U_i, B'_i, k_0) \cdot l = \sum_{\mathbf{B}_{-i} \in k_0} U_i(F(B'_i, \mathbf{B}_{-i})).$$

Some investigation of U_i lears us that this implies

$$E_F(U_i, B'_i, k_0) \cdot l = 2l \cdot l - |\{\mathbf{B}_{-i} \mid F(B'_i, \mathbf{B}_{-i}) \neq \{B_i\}\}| - (2l - 1) \cdot |\{\mathbf{B}_{-i} \mid F(B'_i, \mathbf{B}_{-i}) = S\}|.$$

Since $\{B_i\} \neq S$, we must have that $l = |k_0| \geq 2$ and thus we also have $l \geq |\{\mathbf{B}_{-i} \mid F(B'_i, \mathbf{B}_{-i} \neq \{B_i\})\}|$, thus we continue

$$\begin{aligned} E_F(U_i, B'_i, k_0) \cdot l &\geq 2l \cdot l - l - (2l - 1) \cdot |\{\mathbf{B}_{-i} \mid F(B'_i, \mathbf{B}_{-i}) = S\}| \\ &> 2l \cdot l - (2l - 1) - (2l - 1) \cdot |\{\mathbf{B}_{-i} \mid F(B'_i, \mathbf{B}_{-i}) = S\}| \\ &= 2l \cdot l - (2l - 1) \cdot (|\{\mathbf{B}_{-i} \mid F(B'_i, \mathbf{B}_{-i}) = S\}| + 1). \end{aligned}$$

Now by assumption we have $|\{\mathbf{B}_{-i} \mid F(B'_i, \mathbf{B}_{-i}) = S\}| < |\{\mathbf{B}_{-i} \mid F(B_i, \mathbf{B}_{-i}) = S\}|$, hence we can continue

$$\begin{aligned} &\geq 2l \cdot l - (2l - 1) \cdot |\{\mathbf{B}_{-i} \mid F(B_i, \mathbf{B}_{-i}) = S\}| \\ &\geq 2l \cdot l - |\{\mathbf{B}_{-i} \mid F(B_i, \mathbf{B}_{-i} \neq \{B_i\})\}| - (2l - 1) \cdot |\{\mathbf{B}_{-i} \mid F(B_i, \mathbf{B}_{-i}) = S\}| \\ &= E_F(U_i, B_i, k_0) \cdot l. \end{aligned}$$

Hence $E_F(U_i, B'_i) > E_F(U_i, B_i)$, so F is full-ignorance manipulable by agent i with utility U_i that is top-respecting to U_i. ☺

We can use this characterization to prove that full-ignorance manipulability implies full-knowledge manipulability.

Theorem 18. *If an aggregation rule F is full-ignorance manipulable by agent i with utility U_i top-respecting to B_i, then it is full-knowledge manipulable by agent i on some profile including B_i.*

Proof. Suppose F is full-ignorance manipulable by agent i with utility U_i top-respecting to B_i. Then by Theorem 17, there is an alternative ballot $B_i' \neq B_i$ and some set of ballots $S \neq \{B_i\}$ such that $|\{\mathbf{B}_{-i} \mid F(B_i', \mathbf{B}_{-i}) = S\}| < |\{\mathbf{B}_{-i} \mid F(B_i, \mathbf{B}_{-i}) = S\}|$. Hence we must have that there is some $\min_i$-profile $\mathbf{B}_{-i}$ such that $F(B_i, \mathbf{B}_{-i}) = S \neq \{B_i\}$ and $F(B_i', \mathbf{B}_{-i}) \neq S = F(B_i, \mathbf{B}_{-i})$. Hence Theorem 14 implies that F is manipulable under full knowledge at profile $B_i, \mathbf{B}_{-i}$ by agent i. ☺

This theorem does not hold in the other direction. Consider the following example.

Example 10. Let $n = 3$, $m = 4$ and let

$$F(B_0, B_1, B_2) = \begin{cases} (\mathsf{False}, \mathsf{False}, \mathsf{False}, \mathsf{False}) & \text{if } b_{0,1} = b_{1,1}, \\ (\mathsf{True}, \mathsf{True}, \mathsf{True}, \mathsf{True}) & \text{otherwise.} \end{cases}$$

Now let

$$U_1(B) = \begin{cases} 1 & \text{if } B = (\mathsf{True}, \mathsf{True}, \mathsf{True}, \mathsf{True}) \\ 0 & \text{otherwise.} \end{cases}$$

Clearly agent 1 can manipulate F if she knows $B_{0,1}$. However, under full ignorance, her expected utility for all possible ballots is equal to 0.5. Thus agent 1 cannot manipulate F. ■

The rule in this example is a non-trivial one, since it is neither constant nor a dictatorship. Hence this shows that there are non-trivial rules that cannot be manipulated under full ignorance. This is a very nice result, since this implies that the result of Theorem 15 can be circumvented if only we can make sure that agents have the least information possible about each others preferences. We characterized the class of aggregation rules for which this holds. This characterisation of Theorem 17 does seem very strict. An interresting line of research for the future is to study these rules and to investigate whether there are cases in which these rules are suitable.

3.3 Setting the aggregation procedure

In the previous section we investigated how agents can manipulate aggregation rules by lying about their preferences. Here we assume that the aggregation rule is fixed. However, when voting, often agents first have to agree on an aggregation rule. They probably will not vote on this, since then, what aggregation rule would they use to determine which rule to use? This would create a Droste effect from which they can never escape! So how does this work? For official bodies like parliament, sometimes a voting procedure is established in laws or rules. Or there is some council that determines the procedure.[3] In smaller or less formal groups often it happens that somebody proposes some kind of rule that sounds reasonable and nobody protests. When properly prepared, one can use this to her advantage and have much more influence than by manipulating the rule itself. We will now highlight two ways in which this can be manipulated. When the issues to be voted on are set, agents can manipulate the outcome

[3]This happend for example in Leininger (1993). Here the decision on the voting procedure made all of the difference! For any possible outcome a voting procedure could have been found that would give this outcome.

by suggesting certain rules. However, this is not the only thing that can be manipulated when deciding on a procedure. When the kind of aggregation rule that will be used is set, agents can manipulate by setting the agenda, which is the list of issues to be voted on. We will see that the same rule will give different outcomes for different agendas.

3.3.1 Choosing an aggregation rule

Example 6 - Spending money (continued). Suppose that the family from Example 6 will first decide on an aggregation rule before they vote for what to buy. Consider agent 2, with equally weighted Hamming utility to the ballot represented by integer 8, and with the specific amount of knowledge `exampleKn2`. We have seen before that her best pick for a ballot to cast was different for the priority rule and the distance based rule, namely the ballots represented by 24 and 8 respectively. As an anticipating agent, she can use this information to influence the choice of a rule. Both best ballots give different expected utilities for their corresponding aggregation rules.

```
  > expectedUBinA (i2b 5 24) (uSet (uH (i2b 5 8) 5)) exampleKn2 5 l0 priority
1.6666666
  > expectedUBinA (i2b 5 8) (uSet (uH (i2b 5 8) 5)) exampleKn2 5 l0 dbRule
1.4444444
```

Because the best ballot for the priority rule gives agent 2 a higher expected utility than her best ballot for the distance based rule, she should suggest to use the priority rule. ■

The above example shows that sometimes an agent should prefer a certain rule, to maximize her expected utility. For a given rule, knowledge and utility we can calculate what ballot the agent should cast. This ballot gives her a certain expected utility, that is higher than her expectations when casting any other ballot. We calculate this best expectation.

```
bestExpectation :: Int -> UtilitySet -> Knowledge -> IC -> AR -> Float
bestExpectation m ui kn bc rule = let
    b = head (bestBallots (models m bc) ui kn m bc rule)
  in
    expectedUBinA b ui kn m bc rule
```

To find out which rule is optimal, we can compare the best expectations.

```
  > bestExpectation 5 uExHome
  (noKnowledge 5 3 2 l0) l0 dbRule
240.488
  > bestExpectation 5 uExHome
    (noKnowledge 5 3 2 l0) l0 priority
247.24
```

In this case, where the agent is fully ignorant, she could better suggest the priority rule than the distance based rule, since this gives her a higher best expectation. Note that we assume the agent will

try to manipulate the rule as well. If she would not do that, we should compare the expectation for her true ballot instead.

We implement a function that picks the best rules from a list of possible rules. The output is a list containing the indices of the best rules in the inputlist.

```
bestRules :: Int -> UtilitySet -> Knowledge -> IC -> [AR] -> [Int]
bestRules m ui kn bc rules = maximaBy
    (comparing (\r -> bestExpectation m ui kn bc (rules !! r)))
    [0..(length rules - 1)]
```

Indeed in our example the priority rule is better than the distance based rule.

```
  > bestRules 5 uExHome (noKnowledge 5 3 2 l0) l0 [priority, dbRule]
[[1]]
```

3.3.2 Agenda setting

When deciding on certain issues, first an agenda has to be set. That is; it has to be decided on which issues exactly will be voted. To illustrate this, in Example 1, a decision on the issue "cat and dog" can be reached by majority voting on the issues "cat" and "dog", but also by majority voting on the issue "cat and dog". These are two different agendas that both settle the issue "cat and dog". Note that in this case, the setting of the agenda actually matters for the outcome of the rule. When voting by majority for the separate issues, the family will decide to buy a cat and a dog, however, when voting for only the one issue "cat and dog", the family will not. An agenda setter can use this fact to her advantage. She can set the agenda in such a way that it will affect the collective outcome on certain propositions. This is called *agenda manipulation*.

In this section, we will start with some definitions to make this more precise and we will use these to give some more examples of manipulations. We start with the formal definition of an agenda. Let Ω be a fixed non-empty set of issues, together with a constraint IC. These might be infinite, but for simplicity we will now assume they are finite and $|\Omega| = m$. An agenda A is a subset of Ω. It represents the set of issues that we are voting on. Note that as it is a set, the order does not matter, just as the order of the issues in $\mathcal{I}$ did not matter. However, we do need to be careful if we study rules that are not issue-neutral, as these rules should come together with an ordering on the agenda.

```
type Agenda = [Issue]
```

An Agenda A is said to extend an agenda A' if $A' \subseteq A$.

The reason that we could vote for different issues than the ones we actually want to decide upon, is that when we are dealing with an integrity constraint, often our decision on certain issues is forced by our decision on other ones (e.g. in Example 1, the decision for "cat and dog" is forced by our decisions on the other two issues).

A ballot B that expresses the preferences on an agenda A is called a ballot over A. Two ballots B and

B' over agendas A and A' respectively are consistent when they agree on all issues in $A \cap A'$. Two profiles $\mathbf{B}$, $\mathbf{B}'$ are consistent when for any $i \in \mathcal{N}$, B_i is consistent with B'_i. Two sets S_1, S_2 of ballots are consistent if for any $B \in S_1$ there is a ballot $B' \in S_2$ such that B is consistent with B', and reversely.

```
consistentBallot :: Agenda -> Agenda -> Ballot -> Ballot -> Bool
consistentBallot ag1 ag2 b1 b2 =
  all (\j -> case elemIndex (ag1 !! j) ag2 of
        Nothing -> True
        Just ix -> b1 !! j == b2 !! ix)
  [0..(length ag1-1)]
```

The ballot (True, True) on agenda $\{0, 1\}$ is consistent with the ballot (True, False) on agenda $\{1, 2\}$ since they both agree on issue 1, which is the only element in the intersection of the two agendas.

```
  > consistentBallot [0,1] [1,2] [True, True] [True, False]
True
  > consistentBallot [0,1] [1,2] [True, True] [False, False]
False
```

We can create a set of all rational ballots over some agenda A' that are consistent with a certain ballot B over another agenda A. We call this the set of ballots over A' that correspond to B.

```
corBallots :: Agenda -> Agenda -> Int -> IC -> Ballot -> [Ballot]
corBallots ag1 ag2 m bc b = let
    bs = models (length ag2) (restrictIC ag2 m bc)
  in
    filter (consistentBallot ag1 ag2 b) bs
```

We test what ballots over agenda $\{1, 2\}$ correspond to ballot (True, True) over agenda $\{0, 1\}$.

```
  > corBallots [0,1] [1,2] 3 n0 [True, True]
[[True,True],[True,False]]
```

When B is a ballot over A, we can restrict this ballot to a smaller agenda. Suppose A' is a different agenda. We denote the corresponding ballot over $A \cap A'$ as $B{\restriction_{A'}}$. This is the ballot that holds a vote for every issue in $A \cap A'$, and agrees with ballot B on all of these issues. We can also restrict a set of ballots or a profile, by restricting every ballot in the set or profile. The restriction of profile $\mathbf{B}$ and set S to agenda A is written as $\mathbf{B}{\restriction_A}$ and $S{\restriction_A}$ respectively. Note that two ballots B and B' over agendas A and A' respectively are *consistent* when $B{\restriction_{A'}} = B'{\restriction_A}$.

```
restrictBal :: Agenda -> Agenda -> Ballot -> Ballot
restrictBal ag1 ag2 b1 = head (filter (consistentBallot ag1 ag2 b1)
                          (models m n0)) where
                      m = length ag2
```

```
restrictProfile :: Agenda -> Agenda -> Profile -> Profile
restrictProfile ag1 ag2 = map (restrictBal ag1 ag2)

restrictKnowledge :: Agenda -> Agenda -> Knowledge -> Knowledge
restrictKnowledge ag1 ag2 = map (restrictProfile ag1 ag2 ***
                                 restrictProfile ag1 ag2)
```

A ballot over a smaller agenda can still satisfy the integrity constraint (where the integrity constraint is defined over Ω). For this we restrict an integrity constraint IC over Ω to ballots over A, by saying that ballot B on A satisfies IC if and only if there is a rational ballot B' over Ω that can be restricted to B. The set of rational ballots over A is denoted as $\mathsf{Mod}^A(\mathrm{IC})$.

```
restrictIC :: Agenda -> Int -> IC -> IC
restrictIC ag m bc b =
    any (consistentBallot ag [0..(m-1)] b) (models m bc)
```

Let S be a set of rational ballots on some agenda A. We define

$$\mathcal{E}(S) = \{B \in \mathsf{Mod}(\mathrm{IC}) \mid B \text{ is consistent with some } B' \in S\}.$$

This is the set of ballots over Ω that are consistent with some ballot in S. We write $\mathcal{E}(B)$ for $\mathcal{E}(\{B\})$. We are also interested in the utility for ballots over a specific agenda. Since a ballot might correspond to a set of ballots on Ω, we let the utility on single ballots be the utility of the corresponding set. We let the utility on sets of ballots be the average of the utility on single ballots, assuming a random tie-breaking on the outcome of the rule on the Agenda. We will denote the utility U_i restricted to A as $U_i{\restriction}_A$.

Definition 22. The utility U_i restricted to agenda A is given by

$$U_i{\restriction}_A(S) = \frac{\sum_{B \in S} U_i(\mathcal{E}(\{B\}))}{|S|}$$

We encode this.

```
restrictUtility :: Agenda -> Agenda -> Int ->
                   IC -> UtilitySet -> UtilitySet
restrictUtility ag1 ag2 m bc u bs = let
    singleu b = u (corBallots ag2 ag1 m bc b)
  in
    uSet singleu bs
```

We can restrict the utility `uExHome` from Example 6 to the agenda $\{1, 2\}$. We can use this to calculate the utility for a smaller ballot on the smaller agenda.

```
  > restrictUtility [0..4] [1..2] 5 l0 uExHome [[True,False]]
221.8
```

A set of ballots S over an agenda A *entails* a vote v on an issue $j' \in \Omega$ if every rational ballot B' over Ω that is consistent with some $B \in S$ satisfies $b'_{j'} = v$. An agenda A *settles* an agenda A' if any rational ballot $B \in \mathsf{Mod}^A(\mathrm{IC})$ is consistent with exactly one $B' \in \mathsf{Mod}^{A'}(\mathrm{IC})$ (or equivalently, if for each $B \in \mathsf{Mod}^A(\mathrm{IC})$, $\{B\}$ entails a vote v_j for all $j \in A'$). We encode a function that tests whether one agenda settles another one.

```
settles :: Int -> Agenda -> Agenda -> IC -> Bool
settles m ag1 ag2 bc = let
    l1 = length ag1
    ag2' = nub (ag2 ++ ag1)
    l2 = length ag2'
    bc1 = restrictIC ag1 m bc
    bc2 = restrictIC ag2' m bc
    bs1 = models l1 bc1
    bs2 = models l2 bc2
    f b = length (filter (consistentBallot ag1 ag2' b) bs2) == 1
  in
    all f bs1
```

In Example 1, the issue "cat and dog" is settled by the issues "cat" and "dog".

```
  > settles 3 [0,2] [1] q0
True
```

The *scope* $\bar{A}$ of an agenda A is the largest set A' such that A settles A'. Suppose A' is the agenda of issues that we actually want to make a decision on. We could set the agenda A', however, there might be another agenda that also settles A' which we could vote on. We create a list of all agendas that settle a certain agenda.

```
allAgsForAg :: Int -> Agenda -> IC -> [Agenda]
allAgsForAg m ag' bc = filter (\ag -> settles m ag ag' bc)
              (subsequences [0..(m-1)])
```

Consider Example 1. Suppose the family comes across a deal in which they can buy a cat and a dog together. When deciding whether to take this deal or not, several agendas are possible that settle their question.

```
  > allAgsForAg 3 [1] q0
[[1],[0,1],[0,2],[1,2],[0,1,2]]
```

Of course, agenda `[0,1]` does not make a lot of sense. Why would they vote both for "having a cat and a dog" and for "having a dog" when they only need to decide on the former? The reason that this agenda is somehow strange, is that it is not minimal, that is, there is a subset of the agenda that already settles the issues we want to decide on. We can also consider all minimal agendas that settle our issues.

```
minimals :: [Agenda] -> [Agenda]
minimals ags = minimals' ags ags where
  minimals' ags' [] = ags'
  minimals' ags' (x:xs) = let
      f y = x == y || not (all (`elem` y) x)
    in
      minimals' (filter f ags') xs
```

```
  > minimals (allAgsForAg 3 [1] q0)
[[1],[0,2]]
```

When setting the agenda, we might have certain restrictions on the agenda, for example that the agenda is a minimal settling agenda for a certain set of issues. We might also have other restrictions on the size or complexity of the agenda. For example, agents might protest if they have to vote for a hundred issues, when their problems can also be solved by voting on two issues. To capture this, we consider a set of *feasible* agendas $\mathcal{A} \subseteq \mathcal{P}(\Omega)$. In our examples, we let $\mathcal{A}$ be the set of minimal agendas.

Definition 23. An aggregation system is a family $(F_A)_{A\in\mathcal{A}}$ containing an aggregation rule F_A for each feasible agenda $A \in \mathcal{A}$.

An aggregation system is called collectively rational with respect to an integrity constraint IC if every aggregation rule in the system is collectively rational with respect to the restriction of IC to the agenda.

An example of a collectively rational aggregation system is the system $(DB_A)_{A\in\mathcal{A}}$ such that DB_A is the distance based rule on A. We call this the distance based system.

```
dbOnAg :: Agenda -> AR
dbOnAg ag m bc = let
    bc' = restrictIC ag m bc
    m' = length ag
  in
    dbRule m' bc'
```

We apply the distance based rule on two different agendas for Example 6.

```
  > dbOnAg [0,1,2] 5 l0 exampleC
[[False,True,True]]
  > dbOnAg [1,2] 5 l0 exampleC
[[False,True]]
```

On these different agendas, the distance based rule gives different outcomes for issue 1. This implies that an agent could use this fact to her advantage when setting the agenda.

In Dietrich (2013) definitions for agenda-insensitivity of resolute rules are given. We will now give definitions for non-resolute rules.

Definition 24. A collectively rational aggregation system $(F_A)_{A\in\mathcal{A}}$ is *basically agenda-insensitive* if for any two feasible agendas $A, A' \in \mathcal{A}$, we have for any rational profile $\mathbf{B}$,

$$F_A(\mathbf{B}\restriction_A)\restriction_{A'} = F_{A'}(\mathbf{B}\restriction_{A'})\restriction_A.$$

When F is resolute, this reduces to having that $F_A(\mathbf{B}\restriction_A)$ and $F_{A'}(\mathbf{B}\restriction_{A'})$ must agree on all issues in $A \cap A'$, which is equivalent to the definition of basically agenda-insensitivity given by Dietrich (2013).

Definition 25. A collectively rational aggregation system $(F_A)_{A\in\mathcal{A}}$ is *agenda-insensitive* if for any two feasible agendas $A', A'' \in \mathcal{A}$, we have for any rational profile $\mathbf{B}$, and any $B \in \mathsf{Mod}(\mathrm{IC})$ that

$$\frac{\sum\limits_{\substack{B'\in F_A(\mathbf{B}|_A)\\ \text{s.t. } B\in\mathcal{E}(\{B'\})}} \frac{1}{|\mathcal{E}(\{B'\})|}}{\sum\limits_{\substack{B'\in F_{A'}(\mathbf{B}\restriction_{A'})\\ \text{s.t. } B\in\mathcal{E}(\{B'\})}} \frac{1}{|\mathcal{E}(\{B'\})|}} = \frac{|F_A(\mathbf{B}|_A)|}{|F_{A'}(\mathbf{B}\restriction_{A'})|}$$

When the aggregation system is resolute, this reduces to having for any rational profile $\mathbf{B}$ that

$$\mathcal{E}(F_A(\mathbf{B}|_A)) = \mathcal{E}(F_{A'}(\mathbf{B}\restriction_{A'})),$$

which is then equivalent to the definition of full agenda-insensitivity given in Dietrich (2013).

Definition 26. An agent i with utility U_i is said to be able to *simply manipulate the agenda* for aggregation system $(F_A)_{A\in\mathcal{A}}$ on profile $\mathbf{B}$ if there are $A, A' \in \mathcal{A}$ such that $U_i(F_A(\mathbf{B}\restriction_A)) \neq U_i(F'_A(\mathbf{B}\restriction_{A'}))$.

This means that an agent can simply manipulate the agenda on a certain profile if there are different agendas in the feasible set of agendas such that the outcomes of the corresponding aggregation rules give a different utility.

Example 6 - Spending money (continued). Suppose the family always uses the distance based rule, but an agenda is not set. Suppose $\{0,1,2\}$ and $\{1,2\}$ are both feasible agendas. Consider an agent with utility $uExHome$, we calculate the utilities for the outcome on different agendas.

```
  > restrictUtility [0..4] [0..2] 5 l0 uExHome
  (dbOnAg [0,1,2] 5 l0 exampleC)
102.0
  > restrictUtility [0..4] [1..2] 5 l0 uExHome
  (dbOnAg [0,1,2] 5 l0 exampleC)
100.5
```

We see that agenda $\{0,1,2\}$ is slightly better for this agent than agenda $\{1,2\}$. Hence this agent can simply manipulate the agenda. ■

Theorem 19. *The aggregation system $(F_A)_{A\in\mathcal{A}}$ is agenda insensitive if and only if there is no agent i and utility U_i such that agent i with utility U_i can simply manipulate the agenda on some profile.*

Proof. Suppose $(F_A)_{A\in\mathcal{A}}$ is agenda insensitive. Let i be an agent, U_i a utility function, $\mathbf{B}$ a profile,

$A', A'' \in \mathcal{A}$ agendas. Now we have

$$\begin{aligned} U_i\restriction_A(F_A(\mathbf{B}\restriction_A)) &= \frac{\sum\limits_{B\in F_A(\mathbf{B}\restriction_A)} U_i\restriction_A(B)}{|F_A(\mathbf{B}\restriction_A)|} \\ &= \frac{\sum\limits_{B\in F_A(\mathbf{B}\restriction_A)} U_i(\mathcal{E}(\{B\}))}{|F_A(\mathbf{B}\restriction_A)|} \\ &= \frac{\sum\limits_{B\in F_A(\mathbf{B}\restriction_A)} \frac{\sum\limits_{B'\in\mathcal{E}(\{B\})} U_i(B')}{|\mathcal{E}(\{B\})|}}{|F_A(\mathbf{B}\restriction_A)|} \\ &= \sum_{B\in F_A(\mathbf{B}\restriction_A)} \sum_{B'\in\mathcal{E}(\{B\})} \frac{U_i(B')}{|\mathcal{E}(\{B\})|\cdot|F_A(\mathbf{B}\restriction_A)|} \\ &= \sum_{B'\in\mathsf{Mod}(\mathrm{IC})} U_i(B')\cdot \sum_{\substack{B\in F_A(\mathbf{B}\restriction_A)\\ \text{s.t. } B'\in\mathcal{E}(\{B\})}} \frac{1}{|\mathcal{E}(\{B\})|\cdot|F_A(\mathbf{B}\restriction_A)|}. \end{aligned}$$

And since the system is agenda insensitive, this equals

$$= \sum_{B'\in\mathsf{Mod}(\mathrm{IC})} U_i(B') \sum_{\substack{B\in F_{A'}(\mathbf{B}\restriction_{A'})\\ \text{s.t. } B'\in\mathcal{E}(\{B\})}} \frac{1}{|\mathcal{E}(\{B\})|\cdot|F_{A'}(\mathbf{B}\restriction_{A'})|}$$

which then by similar steps back equals

$$= U_i\restriction_{A'}(F_{A'}(\mathbf{B}\restriction_{A'})).$$

Thus $U_i\restriction_A(F_A(\mathbf{B}\restriction_A)) = U_i\restriction_{A'}(F_{A'}(\mathbf{B}\restriction_{A'}))$, hence agent i cannot manipulate the agenda on profile $\mathbf{B}$. Thus since i, U_i, $\mathbf{B}$ were arbitrary, no agent can simply manipulate the agenda.

Now suppose $(F_A)_{A\in\mathcal{A}}$ is not agenda insensitive, hence there are $A', A'' \in \mathcal{A}$, a profile $\mathbf{B}$, and a ballot $B \in \mathsf{Mod}(\mathrm{IC})$ such that

$$\frac{\sum\limits_{\substack{B'\in F_A(\mathbf{B}|_A)\\ \text{s.t. } B\in\mathcal{E}(\{B'\})}} \frac{1}{|\mathcal{E}(\{B'\})|}}{\sum\limits_{\substack{B'\in F_{A'}(\mathbf{B}\restriction_{A'})\\ \text{s.t. } B\in\mathcal{E}(\{B'\})}} \frac{1}{|\mathcal{E}(\{B'\})|}} \neq \frac{|F_A(\mathbf{B}|_A)|}{|F_{A'}(\mathbf{B}\restriction_{A'})|}.$$

This implies

$$\sum_{\substack{B'\in F_A(\mathbf{B}|_A)\\ \text{s.t. } B\in\mathcal{E}(\{B'\})}} \frac{1}{|\mathcal{E}(\{B'\})|\cdot|F_A(\mathbf{B}|_A)|} \neq \sum_{\substack{B'\in F_{A'}(\mathbf{B}\restriction_{A'})\\ \text{s.t. } B\in\mathcal{E}(\{B'\})}} \frac{1}{|\mathcal{E}(\{B'\})|\cdot|F_{A'}(\mathbf{B}\restriction_{A'})|}.$$

Now let U_i be given by $U_i(B) = 1$ and $U_i(B') = 0$ for any $B' \neq B$. Now we have, using the derivation

made before in this proof, that

$$\begin{aligned} U_i{\restriction_A}(F_A(\mathbf{B}{\restriction_A})) &= \sum_{B' \in Mod(\mathrm{IC})} U_i(B') \sum_{\substack{B'' \in F_A(\mathbf{B}{\restriction_A}) \\ \text{s.t. } B' \in \mathcal{E}(\{B''\})}} \frac{1}{|\mathcal{E}(\{B''\})| \cdot |F_A(\mathbf{B}{\restriction_A})|} \\ &= 1 \cdot \sum_{\substack{B'' \in F_A(\mathbf{B}{\restriction_A}) \\ \text{s.t. } B \in \mathcal{E}(\{B'''\})}} \frac{1}{|\mathcal{E}(\{B''\})| \cdot |F_A(\mathbf{B}{\restriction_A})|} \\ &\neq \sum_{\substack{B'' \in F_{A'}(\mathbf{B}{\restriction_{A'}}) \\ \text{s.t. } B \in \mathcal{E}(\{B''\})}} \frac{1}{|\mathcal{E}(\{B''\})|} \cdot |F_{A'}(\mathbf{B}{\restriction_{A'}})| \\ &= U_i{\restriction_{A'}}(F_{A'}(\mathbf{B}{\restriction_{A'}})) \end{aligned}$$

Hence agent i can simply manipulate the agenda on profile $\mathbf{B}$. ☺

Combining agenda manipulation and rule manipulation

The above definition assumes that an agent will not lie about her preferences and chooses an agenda that gives an optimal outcome for her true preferences. However, if an agent is willing to manipulate the agenda, she will most likely have no problem with manipulating the aggregation rule as well. Therefore it is much more likely that she will not specifically want an agenda such that the outcome for her true preferences will be maximal, but that she would like to have an agenda that will maximize her utility considering that she would manipulate the rule after that.

Definition 27. An agent i with utility U_i is said to be able to *manipulate the agenda* for aggregation system $(F_A)_{A \in \mathcal{A}}$ on profile $\mathbf{B}$ if there are $A, A' \in \mathcal{A}$, and a ballot B_i' such that $U_i(F_A((B_i', \mathbf{B}_{-i}){\restriction_A})) > U_i(F_A'((B_i'', \mathbf{B}_{-i}){\restriction_{A'}})) \quad \forall B_i'' \in \mathsf{Mod}(\mathrm{IC})$.

This means that an agent can manipulate the agenda on a certain profile $\mathbf{B}$ if there are two agendas $A, A' \in \mathcal{A}$, and some ballot B_i' such that the utility of the outcome of F_A when casting B_i' is higher than all utilities of the possible outcomes for $F_{A'}$. The difference with a simple manipulation is that the ballot of the manipulating agent is not fixed. The agent is expected to manipulate the agenda in such a way that she can cast a ballot that would give her an optimal outcome. Note that it is possible that an agenda is simply manipulable but not manipulable by some agent. However this implies that the rules in the aggregation system are manipulable for this agent.

Agenda setting under partial knowledge

We investigated when agenda manipulation would be possible for agents having full knowledge about the preferences of the others. We can now easily extend this to the case in which we have only partial knowledge on the preferences of the other agents.

Definition 28. An agent i with utility U_i is said to be able to *manipulate the agenda* under knowledge k for aggregation system $(F_A)_{A \in \mathcal{A}}$ if there are $A, A' \in \mathcal{A}$, and a ballot B_i' such that $E_{F_A}(U_i{\restriction_A}, B_i'{\restriction_A}, k) > E_{F_A}(U_i{\restriction_A}, B_i''{\restriction_A}, k) \quad \forall B_i'' \in \mathsf{Mod}(\mathrm{IC})$.

This means that an agent can manipulate the agenda under knowledge k if there are two agendas $A, A' \in \mathcal{A}$, and some ballot B_i' such that the expected utility for F_A when casting B_i' is higher than the expected utility for $F_{A'}$ for any cast ballot B_i''.

In the section on manipulation of the aggregation rule we encoded a function that enables us to calculate an expected utility given the rule, our knowledge, the agenda, and our own ballot. We used this to find a maximal expected utility (given by the function `bestExpectation`). If we calculate this best expectation for multiple agendas, we can find out which agenda maximizes this best expectation. In this way we combine agenda manipulation with manipulation of the aggregation rule.

We encode this for the distance based system. We assume that $\mathcal{A}$ is the full set of agendas that settle a given agenda. Now we can pick the best agendas for this system, given our utility, integrity constraint and knowledge over Ω. For each agenda, we calculate what ballot gives us the best expected utility. We choose the agendas for which this is maximal.

```
bestAgendasDB :: Int -> Agenda -> IC -> UtilitySet ->
                  Knowledge -> [Agenda]
bestAgendasDB m ag' bc ui kn = let
    ags = allAgsForAg m ag' bc
    ag1 = [0..(m-1)]
    f ag = bestExpectation (length ag) (restrictUtility ag1 ag m bc ui)
          (restrictKnowledge ag1 ag kn)
          (restrictIC ag m bc) (dbOnAg ag)
  in
    maximaBy (comparing f) ags
```

We apply this to some of our examples. We calculate an optimal agenda, given our knowledge.

```
  > bestAgendasDB 5 [0,1] l0 uExHome (noKnowledge 5 3 2 l0)
[[0,1,2]]
  > bestAgendasDB 3 [0] q0 (uSet (uHW w1 (i2b 3 7) 3)) (noKnowledge 3 3 1 q0)
[[0,2]]
```

Thus, in Example 1, under full ignorance an agent with utility `uExHome` should put the issues "cat" and "dog" on the agenda to obtain an optimal decision on the issue "dog". In Example 6, to settle the first two issues the best agenda for an agent with no knowledge and utility `uSet (uHW w1 (i2b 3 7) 3)` is the agenda consisting of the first three issues.

3.4 Summary

Utility functions can capture the satisfaction that agents will have with the outcome of the aggregation procedure. Examples show that we should not make too many assumptions on utility functions. Using these utilities, we generalized some definitions and results on manipulability. Agents can manipulate a rule onder a certain amount of knowledge, by trying to maximize their expected utility. They can also influence the choice of a rule or the setting of the agenda in order to obtain a better outcome for themselves. Manipulability is almost unavoidable. Even full ignorance often does not ensure non-manipulability.

4 | Quantifying manipulability of aggregation rules

As we have seen, with or without knowledge, possible manipulability is almost unavoidable. However, these results say nothing about the **frequency** and **impact** of the possible manipulations. We would like to be able to quantify how susceptible a rule is to manipulability. For this we consider several approaches. First we look at manipulability under full knowledge. All code used in this section can be found in Appendix B. In this section we will apply all techniques to Example 6 (for some different n), however, the implementation can be used to calculate manipulative powers of rules for several different aggregation problems.

4.1 Basic manipulative power

A natural approach for quantifying manipulability of social choice functions was taken in Friedgut et al. (2011). In this method, any social choice function is associated with a manipulative power, which is given by the percentage of the profiles in which a manipulation is possible. When we assume every agent has full knowledge, this is quite straightforward, since ballots give the order of preference of the alternatives. This is all the information we need to be able to calculate whether certain agents can manipulate under full knowledge. However under partial knowledge, we would need to know the full utility function of the agents and not just the preference order.

For binary aggregation rules it is somewhat less straightforward, since we only know the judgement of the agents on the separate issues, hence we do not know their preference ordering on possible outcomes. This said, we can calculate the percentage of the profiles in which a possible manipulation might happen, that is, agents can force another outcome that they might prefer.[1]

Definition 29. The *basic manipulative power* of an aggregation rule F (for a fixed m, n, IC) for an $i \in \mathcal{N}$ with respect to a class of utility functions $\mathcal{C}$ is the percentage of rational profiles on which there exists a utility function $U_i \in \mathcal{C}$ such that agent i can manipulate F. The basic manipulative power of an aggregation rule F with respect to a class of utility functions $\mathcal{C}$ is the average of the basic manipulative powers for F with respect to $\mathcal{C}$ for all $i \in \mathcal{N}$.

We will investigate the manipulative power of certain aggregation rules for the class of top-respecting utility functions. We use the characterization of manipulability we proved in Theorem 14.[2]

[1] Clearly this definition of manipulative power relies on the assumption that every rational profile might occur with the same probability. If this is not the case, we could use our prior on the probability distribution of the profiles, to obtain a weighted version of the manipulative power.

[2] Note that this characterises all cases in which an agent might have a utility which allows her to manipulate. Since

In Appendix B a function can be found that calculates the manipulative power of a given agent. For any anonymous aggregation rule, this percentage will be equal for all agents, hence the manipulative power of the rule will equal the manipulative power of an arbitrary agent for that rule. We compare some rules for Example 6, in the case of five family members, for this measure of manipulability.

Rule	Percentage
Priority	69.8
Distance based	70.3
Average voter	68.0
Maximin	94.6
Least squares	85.2

Table 4.1: Basic manipulative powers of different rules for $m = 5$, $n = 5$, integrity constraint l_0, for top-respecting agents

These manipulative powers seem very high. This has several reasons. As noted before, we measure only the cases in which an agent could manipulate for some utility function. For a lot of utilities she might not be able to manipulate. Also this measure is for 5 agents. When the number of agents increases, individual agents have less influence on the outcome, hence these numbers will decrease for higher n.

4.2 Manipulative power

Above we calculated the percentage of the profiles in which an agent might have an incentive to manipulate. However, whether an agent can actually manipulate depends on the agent's utility function. Now we will present a method that does not only take into account the number of cases in which manipulation might be possible, but also considers the likeliness and profitability of the manipulation. Given a fixed utility function, we determine the average win in utility, for a manipulating agent as a percentage of the average utility.

Definition 30. The *manipulative power* of an aggregation rule F (for a fixed m, n, IC) for an $i \in \mathcal{N}$ with respect to a utility function U_i under knowledge k is

$$\frac{\max_{B \in \mathsf{Mod(IC)}} E_F(U_i, B, k) - \max_{B \in \mathsf{Top}(U_i)} E_F(U_i, B, k)}{\frac{\sum_{B \in \mathsf{Mod(IC)}} U_i(B)}{|U_i(B)|}}$$

The manipulative power of an aggregation rule F (for a fixed m, n, IC) for an $i \in \mathcal{N}$ with respect to a class of utility functions $\mathcal{C}$ is the expected manipulative power of F for a random $U_i \in \mathcal{C}$.

Clearly we can take the average over all i to obtain a measure for the manipulability of the rule. However, if the rule is anonymous this does not need to be done.

4.2.1 Under full knowledge

We can try this for full knowledge for a specific weighted Hamming utility we used before.

```
> manipMeas 5 3 1 (uSet (uHW w2 (i2b 5 8) 5)) l0 dbRule
```

there might also be a lot of utilities that do not allow for manipulation, this method only provides us with some worst case scenario.

```
1.7793635e-2
  > manipMeas 5 3 1 (uSet (uHW w2 (i2b 5 8) 5)) l0 priority
9.708025e-2
```

These outcomes represent the expected win in utility when manipulating for this specific utility function. For this, both the distance based rule and the priority rule are not very easy to manipulate (2% and 10% average win in utility respectively). The reason that these numbers are quite small might be that the utility is a (weighted) Hamming utility. Later we will see that for other utilities, these numbers might be much higher.

We are not only interested in the manipulability for a specific utility, but also in the manipulability for a class of utilities. To study that, we will now consider random utilities. We determine a random average respecting utility function by assigning to each ballot a random number and we take the corresponding average respecting utility. We can apply our function to a large number of random utility functions, the average of this will provide us with a more general quantification of the manipulability of the rule under average respecting functions. We compare the manipulability of the priority rule and the distance based rule. The results of this can be found in Table 4.2.2. Here we also compare the two rules for random average respecting weighted Hamming utilities, since these intuitively describe the most rational utilities. To obtain a random average respecting weighted Hamming utility we assign random weights.

4.2.2 Under full ignorance

If we assume that agents are fully ignorant about the preferences of other agents, we would like to know how manipulable our rules are for agents that do not have any knowledge about the preferences of the other agents. With this assumption, the percentage of cases on which manipulation might happen is not very interesting, since this will just equal the percentage of profiles in which the agent can actually influence the outcome. Therefore we once again consider the profitability of manipulation. Here we calculate the expected win in utility for manipulation. We divide by the average utility. We compare the priority rule and the distance based rule.

Utility	Knowledge	Priority rule	Distance based rule
Random avg. resp.	Full	42.2	41.9
Random avg. resp.	No	8.0	7.8
Random avg. resp. Weighted Hamming	Full	2.3	1.4
Random avg. resp. Weighted Hamming	No	2.4	0.0

Table 4.2: Manipulative powers for $m = 5$, $n = 3$, IC $= l_0$ for different classes of utility functions and different amounts of knowledge. Average over 100 random utilities.

We see that the distance based rule is somewhat less manipulable than the priority rule for all measures of manipulability.

4.2.3 Worst-case scenario

In the above we looked at the average profitability of manipulating. Now we will look at the peaks. Can manipulations be really bad? How much utility could somebody win? We measure the expected percentual win for a large number of random utilities and we take the maximal one.

```
  > testMWC Full 5 3 0 10 100
(39.481613,39.45482)
  > testMWC No 5 3 0 10 100
(6.1366763,6.011521)
```

Even when the average percentual win might not be very high, with full knowledge it might happen that an agent can increase her satsifaction with the outcome 39-fold for both the priority and the distance based rule. Without any knowledge, this number is lower, however, for these 100 random utilities, still one agent could increase expected utility 6-fold by manipulating. Of course this is not an absolute maximum, since we took the maximum of a certain number of outcomes. However this does show that even when average manipulations do not seem very problematic, on occasion manipulations can be very lucrative.

4.3 Summary

The standard approach for quantifying manipulability of aggregation rules lacks to take into account the profitability of a manipulation. Using utility functions, we can calculate the manipulative power of a certain rule for a specific utility function, but also for a class of utility functions. This can be used to compare rules for certain situations and to determine which rule is more suitable.

5 | Conclusions and suggestions for future research

Summary and conclusions

In this thesis we studied binary aggregation with integrity constraints. We extended a functional implementation which we used to study aggregation rules and the manipulability of these rules. We suggested a new rule, namely the priority rule. This has some advantages over the premise-based rule, for example that this will also work if there is no possible feasible set of premises. Another suggested rule, the least squares rule, is a rule that we can use in situations in which minorities should have a say as well. The priority rule and the distance based rule have the nice property that they are majority respecting.

We can translate a preference aggregation problem to a binary aggregation problem with a specific integrity constraint. The implementation in this thesis uses this translation to also deal with social choice functions.

Utility functions can capture the satisfaction that agents will have with the outcome of the aggregation procedure. Examples show that we should not make too many assumptions on utility functions. Using these utilities, we generalized some definitions and results on manipulability. Agents can manipulate a rule onder a certain amount of knowledge, by trying to maximize their expected utility. They can also influence the choice of a rule or the setting of the agenda and combine these to obtain a better outcome for themselves. One might hope that with less than perfect knowledge, we could prevent manipulations. A main conclusion of this thesis is that this is true in some nontrivial cases, however not in many. Also, our simulations suggest that it pays off to know the preferences of the other agents.

We investigated how manipulable certain aggregation rules are. Quantification of manipulability of aggregation rules is usually calculated as the percentage of the profiles in which there is an agent that could manipulate. We suggested a different method that also takes into account how much agents can win by manipulating. This method can be used to determine which aggregation rules we should use in different cases. We also provided a measure for manipulability under full ignorance. In cases where we know agents have little knowledge, this measure is much more useful than the standard measure for manipulability.

From this thesis a lot of suggestions and possibilities for further research arise. We will discuss some of them below.

Possible future research

Improve efficiency of the implementation

Our implementation uses the full profile. Most aggregation rules can already be applied on a "compressed profile", that is, we only need to know the ballots that appear in the profile and the number of times they appear. For large n, this would increase the efficiency of the implementation. The implementation could be adapted and improved to make it suitable for more simulations.

Manipulation with non identically distributed beliefs

We only considered identically distributed beliefs. That is, an agent has a belief about what the ballots of the other agents could be, and her ignorance about the remaining possibilities is given by a uniform distribution. For example, she might know that her neighbour will vote $True$ for issue 1, therefore she knows that all profiles in which her neighbour votes $False$ for issue 1 are not possible. Thus, the knowledge of an agent consists of a list of possible partial profiles. However, in real life it might be that an agent does not only have certain knowledge about ballots of other agents, but also has beliefs. For example she might think it is highly likely that some agent casts some vote. We can easily extend our approach by adding probabilities to our knowledge. In this case, knowledge (or belief) of an agent can be represented by a probability function mapping each (rational) profile to a probability that reflects how likely the agent thinks this profile is the true profile. Using this, we can calculate a more realistic expected utility and thus a better manipulation. This can help us in a similar way for agenda manipulation.

Manipulation under full ignorance

We characterized the class of aggregation rules that are not manupulable under full ignorance in Theorem 17. This characterisation does seem strict. A possibility for future research is to study these rules and to investigate whether there are cases in which these rules are suitable.

More results on combined agenda, rule choice and rule manipulation

We provided an implementation for combined agenda and rule manipulation, however we did not give any characterization results for this. When are these kinds of manipulations possible, what do the set of feasible agendas and the aggregation system need to satisfy for such manipulations to exist? Another interresting subject to focus on is the combination of all possible manipulations. We provided an implementation that combined choosing a rule and manipulating it. How would one choose the best aggregation system?

Quantification of manipulation rules

In this thesis we investigated a specific example. The implementations provided can be used to analyse other examples and to study the manipulability of several rules in general (i.e. not for a specific n, m, IC).

Manipulative power with prior on profile distribution

Similarly to the previous suggestion, we could use our prior on the probability distribution of profiles to determine which rules are more and which are less manipulable. When quantifying manipulability, in this thesis we took the average over all possible profiles, assuming they are all equally likely. When we use our prior to calculate a weighted average, this could give us a much better estimate of the likeliness and profitability of possible manipulations.

Towards analyzing aggregation as a Bayesian game

In this thesis we have assumed that the only knowledge agents can have is whether some profiles are possible or not. However, one could also have knowledge about the probability on a certain profile, about the utilities of individual agents, the knowledge of individual agents, and whether the agents are also manipulating or not. In this case, the voting procedure can be seen as a game with incomplete knowledge, a so called Bayesian game. Indeed, it would be fascinating to continue the present study using tools from the theory of Bayesian games.

Bibliography

Brams, S., Kilgour, D., and Sanver, R. (2007). A minimax procedure for electing committees. *Public Choice*, 132(3):401–420.

Copeland, A. (1951). A 'reasonable' social welfare function. In *Seminar on Mathematics in Social Sciences*. University of Michigan.

Dietrich, F. (2006). Judgment aggregation: (im)possibility theorems. *Journal of Economic Theory*, 126(1):286–298.

Dietrich, F. (2013). Judgment aggregation and agenda manipulation. Technical report, University Library of Munich, Germany.

Dietrich, F. and List, C. (2007a). Judgment aggregation by quota rules. *Journal of Theoretical Politics*, 19(4):391–424.

Dietrich, F. and List, C. (2007b). Strategy-proof judgment aggregation. *Political Economy and Public Policy Series*.

Doets, K. and van Eijck, J. (2012). *The Haskell Road to Logic, Maths and Programming, Second Edition*, volume 4 of *Texts in Computing*. College Publications, London. First Edition: 2004.

Eijck, J. v. (2016). Aggregation of preferences. `http://homepages.cwi.nl/~jve/papers/16/aggregation/Aggregation.html`.

Endriss, U. and Grandi, U. (2014). Binary aggregation by selection of the most representative voter. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 668–674. AAAI Press.

Friedgut, E., Kalai, G., Keller, N., and Nisan, N. (2011). A quantitative version of the Gibbard-Satterthwaite theorem for three alternatives. *CoRR*, abs/1105.5129.

Grandi, U. (2012). *Binary aggregation with integrity constraints*. PhD thesis, Institute for Logic, Language and Computation.

Grandi, U. and Endriss, U. (2011a). Binary aggregation with integrity constraints. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume One*, IJCAI'11, pages 204–209. AAAI Press.

Grandi, U. and Endriss, U. (2011b). Binary aggregation with integrity constraints. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*.

Grandi, U. and Endriss, U. (2013). Lifting integrity constraints in binary aggregation. *Artificial Intelligence*, 199-200:45–66.

Khan, A. (2014). Grokking functional programming.

Knuth, D. (1992). *Literate Programming*. CSLI Lecture Notes, no. 27. CSLI, Stanford.

Lang, J., Pigozzi, G., Slavkovik, M., and van der Torre, L. (2011). Judgment aggregation rules based on minimization. In *Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK XIII, pages 238–246, New York, NY, USA. ACM.

Lang, J., Pigozzi, G., Slavkovik, M., van der Torre, L. W. N., and Vesic, S. (2015). Majority-preserving judgment aggregation rules. *CoRR*, abs/1502.05888.

Leininger, W. (1993). The fatal vote: Berlin versus Bonn. *FinanzArchiv / Public Finance Analysis*, 50(1):1–20.

List, C. (2002a). A model of path-dependence in decisions over multiple propositions. Economics Papers 2002-W15, Economics Group, Nuffield College, University of Oxford.

List, C. (2002b). A model of path-dependence in decisions over multiple propositions. Economics Papers 2002-W15, Economics Group, Nuffield College, University of Oxford.

List, C. (2012). The theory of judgment aggregation: An introductory review. *Synthese*, pages 179–207.

List, C. and Pettit, P. (2002). Aggregating sets of judgments: An impossibility result. *Economics and Philosophy*, 18(01):89–110.

List, C. and Puppe, C. (2009). Judgment aggregation: A survey. In Anand, P., Pattanaik, P., and Puppe, C., editors, *Handbook of Rational and Social Choice*. Oxford University Press.

Miller, M. and Osherson, D. (2009). Methods for distance-based judgement aggregation. *Social Choice and Welfare*, 32(4):575–601.

Miller, M. K. and Osherson, D. (2008). Methods for distance-based judgment aggregation. *Social Choice and Welfare*, 32(4):575.

O'Sullivan, B., Goerzen, J., and Stewart, D. (2009). *Real World Haskell*. O'Reilly.

Peterson, J. and Hammond, K. (1997). Report on the programming language Haskell,Version 1.4. Available from the Haskell homepage: `http://www.haskell.org`.

Pigozzi, G. (2006). Belief merging and the discursive dilemma: an argument-based account to paradoxes of judgment aggregation. *Synthese*, 152(2):285–298.

Taylor, A. D. (2005). *Social Choice and the Mathematics of Manipulation*. Cambridge University Press.

Terzopoulou, Z. (2017). Manipulating the manipulators: Richer models of strategic behavior in judgment aggregation. Master's thesis, ILLC, University of Amsterdam.

Zwicker, W. S. (2016). Introduction to the theory of voting. In *Handbook of Computational Social Choice*, chapter 2.

Appendices

A | Some useful functions

A.1 Functions that are not specific to aggregation

```
module MyBasics where
import Data.List
```

We start with some basic useful functions. We use the following notation for material implication.

```
infix 1 -->
(-->) :: Bool -> Bool -> Bool
x --> y = not x || y
```

We use an assertion wrapper for writing spec checking code.

```
assert :: (a -> b -> Bool) -> (a -> b -> String) -> (a -> b) -> a -> b
assert p warning f x = let y = f x in
  if p x y then y else error (warning x y)
```

We can convert a binary number to a decimal integer. We represent the binary number by a list of zeroes and ones.

```
bin2int :: [Bool] -> Int
bin2int = bin . reverse where
  bin []  = 0
  bin [False] = 0
  bin [True] = 1
  bin (False:bs) = 2 * bin bs
  bin (True:bs) = 2 * bin bs + 1
```

We can convert an integer back to a list of bits.

```
int2bin :: Int -> [Bool]
int2bin j = reverse (int2bin' j) where
  int2bin' 0 = []
  int2bin' 1 = [True]
  int2bin' n = odd n : int2bin' (div n 2)
```

We make functions `minimaBy` and `maximaBy`, that are like `minimumBy` and `maximumBy`, but give a list of all elements that are minimal or maximal respectively.

```
groupOrder :: (a -> a -> Ordering) -> [a] ->
              ([a] -> [a] -> Ordering, [[a]])
groupOrder f xs = let
  g a b = f a b == EQ
  grouped = groupBy g (sortBy f xs)
  h ys1 ys2 = f (head ys1) (head ys2)
  in (h, grouped)

minimaBy :: (a -> a -> Ordering) -> [a] -> [a]
minimaBy f xs = let
    (h, grouped) = groupOrder f xs
  in minimumBy h grouped

maximaBy :: (a -> a -> Ordering) -> [a] -> [a]
maximaBy f xs = let
    (h, grouped) = groupOrder f xs
  in maximumBy h grouped
```

We write a function to get user input.

```
get :: String -> IO String
get str = do putStr str
             getLine
```

A.2 Functions that are specific to aggregation

```
module AgHelpFun where
import Aggregation
import Control.Monad
```

We can create a list of all possible profiles with m issues and n agents.

```
allProfiles :: Int -> Int -> IC -> [Profile]
allProfiles m n bc = replicateM n (models m bc)
```

We make a general function that sums integer values over a profile

```
profileSum :: Num a => (Ballot -> a) -> Profile -> a
profileSum fn profile = sum (map fn profile)
```

We implement a changeOnIss function that changes a ballot on a certain issue and a changeBal function that changes a ballot in a profile.

```
changeOnIss :: (Issue,Bool) -> Ballot -> Ballot
changeOnIss (i,b) ballot = let
  pairs = zip [0..] ballot
  f (j,e) = if i==j then (i,b) else (j,e)
  newpairs = map f pairs
 in
  map snd newpairs

changeBal :: Profile -> Agent -> Ballot -> Profile
changeBal prof i b = take i prof ++ [b] ++ drop (i+1) prof
```

Quota voting on an issue

Let's assume we have an odd number of voters. Then majority voting on one issue can be modeled as a map from profiles to profiles, where the minority changes its vote to the vote of the majority. To generalize this we will first implement quota voting for any quota. We distinguish weak and strict majorities.

```
data Quotkind = Strict | Weak
                    deriving(Eq)
quota :: Quotkind -> Float -> Int -> Issue -> Profile -> Bool
quota kind qu _ i profile = let
   n = length profile
   q = fromIntegral n * qu
   decision | kind == Weak =
               fromIntegral (length (coalition i profile)) >= q
            | otherwise =
               fromIntegral (length (coalition i profile)) > q
 in
```

```
    decision
```

Majority voting can be done by using a quota of 0.5.

```
majority :: Quotkind -> Int -> Issue -> Profile -> Bool
majority kind = quota kind 0.5
```

We encode an error for updating knowledge.

```
checkIndices  :: (Int,Int,Agent,Agent,Issue) -> [String]
checkIndices (m,n,j,a,i) =
  ["Agent index too large" | a >= n]
  ++ ["Agent index too small" | a < 0]
  ++ ["Cannot update knowledge about self" | a == j]
  ++ ["Issue index too large" | i >= m]
  ++ ["Issue index too small" | i < 0]
```

A.3 Representative voter rules

```
module MoreRules where
import Aggregation
import AgHelpFun
import Rules
import Data.List
```

Average-voter rule

Definition 31. The *average-voter rule* is the aggregation rule that selects from a profile the individual ballots that mimimise the Hamming distance to the profile.

```
avRule :: AR
avRule _ _ profile = let
    f x = profileSum (hamming x) profile
    mindist z = f z == minimum (map f profile)
  in
    nub (filter mindist profile)
```

Example 11. The profile in Table 11 was also used as an example in Endriss and Grandi (2014). This coincides with the profile `exampleB`.

Issue:	0	1	2	3	4	5
1 voter:	⊤	⊥	⊥	⊥	⊥	⊥
10 voters:	⊥	⊤	⊤	⊥	⊥	⊥
10 voters:	⊥	⊥	⊥	⊤	⊤	⊤

Table A.1: Consider $m = 6$ issues and $n = 21$ voters.

```
exampleB :: ProfInt
exampleB = [32,24,24,24,24,24,24,24,24,24,7,7,7,7,7,7,7,7,7]
```

For this example we obtain the following majority winner.

```
  > majResolve Strict 6 n0 exampleB
[[False,False,False,False,False,False]]
```

■

For Example 11, we get the following average voter outcome.

```
  > avRule 6 n0 exampleB
[[False,True,True,False,False,False]]
```

Majority-voter rule

Definition 32. The *majority-voter rule* is the aggregation rule that selects from a profile the individual ballots that mimimise the Hamming distance to one of the majority outcomes.

```
mvRule :: AR
mvRule m _ profile = let
    f x = minimum (map (hamming x)  (majResolveAll m n0 profile))
    mindist z = f z == minimum (map f profile)
  in
    filter mindist profile
```

For Example 11, we get the following majority-voter outcome.

```
  > mvRule 6 n0 exampleB
[[True,False,False,False,False,False]]
```

Ranked-voter rule

For the ranked-voter rule we first need some more notation.

Definition 33. For a given profile $\mathbf{B}$, the majority strength $MS^{\mathbf{B}}(j)$ of an issue j is given by

$$MS^{\mathbf{B}}(j) = \max\{|N^{\mathbf{B}}_{j:0}|, |N^{\mathbf{B}}_{j:1}|\}$$

```
majStrengt :: Profile -> Issue -> Int
majStrengt profile i = let
    numb1 = length (coalition i profile)
    n = length profile
  in
    maximum [numb1, n - numb1]
```

This induces an ordering $\succ^{\mathbf{B}}_{\tau}$ on issues, with ties broken using a permutation $\tau : \mathcal{I} \to \mathcal{I}$.

```
allPerm :: Int -> [[Issue]]
allPerm m = permutations [0..(m - 1)]

sortIssues:: [Issue] -> Profile -> Issue -> Issue -> Ordering
sortIssues perm profile i j = let
    stronger = compare (majStrengt profile i)
                       (majStrengt profile j)
    higherorder = compare (elemIndex i perm) (elemIndex j perm)
  in
    case stronger of
      EQ -> higherorder
      LT -> LT
      GT -> GT
```

Now we fix a $B^{Maj} \in Maj(B)$ (so B^{Maj} is the weak or strict majority outcome). Then we define the function $\ell^{\mathbf{B}}_{\tau} : I \to 0, 1$ inductively on the ordering $\succ^{\mathbf{B}}_{\tau}$ as follows

$$\ell^{\mathbf{B}}_{\tau}(j) = \begin{cases} b^{Maj}_j & \text{if } \exists i \in \mathcal{N} \text{ such that } b^{Maj}_j = b_{i,j} \text{ and } \forall k \succ^{\mathbf{B}}_{\tau} j, b_{i,k} = \ell^{\mathbf{B}}_{\tau}(k) \\ 1 - b^{Maj}_j & \text{otherwise.} \end{cases}$$

We have that $\ell^{\mathbf{B}}_{\tau}$ defines a ballot. We can also implement this.

```
ltaub :: Quotkind -> Int -> Profile -> [Issue] -> Ballot
ltaub kind m profile perm = let
    beginBallot =  head profile
    issues = sortBy (sortIssues perm profile) [0..(m-1)]
    agreeto j b1 b2 = all (agree b1 b2) (take (j+1) issues)
    existsB b j = any (agreeto j b) profile
    majB = head (majResolve kind m n0 profile)
    lpart ballot j
```

```
      | j == m = ballot
      | existsB (changeOnIss (issues!!j, majB!!(issues!!j))
            ballot) j =
          lpart (changeOnIss (issues!!j, majB!!(issues!!j))
            ballot) (j+1)
      | otherwise =
          lpart (changeOnIss (issues!!j, not (majB!!(issues!!j)))
          ballot) (j+1)
  in
    lpart beginBallot 0
```

Definition 34. A *ranked-voter rule* is an aggregation rule that selects the individual ballots $\ell_\tau^{\mathbf{B}}$ (for either weak or strict majority outcomes) such that τ is some permutation on $\mathcal{I}$.

```
rvRule :: Quotkind -> AR
rvRule kind m _ profile = nub (map (ltaub kind m profile) (allPerm m))
```

For Example 11, we get the following strict ranked-voter outcome.

```
  > rvRule Strict 6 n0 exampleB
[[True,False,False,False,False,False]]
```

We see that for this example, it coincides with the majority-voter, however this is not always the case.

B | Quantifying manipulation functions

Here you can find the code used in section 4.

```
module Quantify where
import Manipulation
import Rules
import AgHelpFun
import Aggregation
import System.Random
import Control.Monad
import Data.List
import Control.Arrow
```

We first encode whether a certain agent might be able to manipulate at a certain profile.

```
manipProfTR :: Int -> Int -> IC -> AR -> Profile -> Bool
manipProfTR m i bc rule prof = let
   bs = models m bc
   x = prof !! i
   u = rule m bc prof
   (y1,y2) = splitProfile i prof
   in u /= [x] &&
      any (\z -> u /= rule m bc (y1 ++ [z] ++ y2))
      bs
```

We can use this to calculate the percentage of profiles on which a certain agent can manipulate. For this we use the characterization of Theorem 14.

```
manipMeasTR :: Int -> Int -> Int -> IC -> AR -> Float
manipMeasTR m n i bc rule = let
   profs = allProfiles m n bc
   manipks = filter (manipProfTR m i bc rule) profs
```

```
    in fromIntegral (length manipks) /
        fromIntegral (length profs)
```

We can also measure only the cases in which a closeness-respecting agent might be able to manipulate. For this we use the characterization of Theorem 12.

```
manipMeasCR :: Int -> Int -> Int -> IC -> AR -> Float
manipMeasCR m n i bc rule = let
  bs = models m bc
  ks = map (\x -> ( x !! i, rule m bc x,
            splitProfile i x))
            (allProfiles m n bc)
  manipl (x, u, (y1,y2)) = any (\z -> ([0..(m-1)] \\
                      agreeSet (head u) x) `intersect`
                      agreeSet x
                      (head (rule m bc (y1 ++ [z] ++ y2)))  /= [])
                            bs
  manipks = filter manipl ks
  in fromIntegral (length manipks) /
      fromIntegral (length ks)
```

Given a fixed utility function, we determine the average win in utility, for a minpulating agent as a percentage of the average utility.

```
manipMeas :: Int -> Int -> Int -> UtilitySet -> IC ->
             AR -> Float
manipMeas m n i ui bc rule = let
  bs = models m bc
  avgU = sum (map (uSingle ui) bs) / fromIntegral (length bs)
  ps = [(p1,p2)| p1 <- replicateM i bs, p2 <- replicateM (n-i-1) bs]
  manipl p = let
    a = maximum (map (\b2 -> ui (
        rule m bc (fst p ++ [b2] ++ snd p ))) bs)
    b = maximum (map (\b1 -> ui (
        rule m bc (fst p ++ [b1] ++ snd p )))
                  (truePrefs m (uSingle ui) bc))
    in  a - b
  manipks = sum (map manipl ps)
  in (manipks /
      fromIntegral (length ps)) / avgU
```

We can use this to run a simulation with a large number of random utilities to quantify the manipulability of aggregation rules. We will later do this without any knowledge as well, hence we will use the following data type.

```
data KnKind = Full | No
                deriving (Eq,Show)
```

To be able to determine which rule is least manipulable for a fixed constraint, m and n, we can generate random utility functions and determine the average win in utility for multiple rules. Here we will compare DB and PR. We take an average over *num* random utility functions.

```
testWithUi :: KnKind -> Int -> Int -> Int -> IC -> Int ->
              [Float] -> (Float,Float)
testWithUi _ _ _ _ _ 0 _ = (0,0)
testWithUi kind m n i bc num uinum =
  let
  ui b = uinum !! b2i b
  meas | kind == Full = manipMeas
       | otherwise = manipMeas0k
  in ((+) (meas m n i (uSet ui) bc priority)
    *** (+) (meas m n i (uSet ui) bc dbRule))
    (testWithUi kind m n i bc (num - 1) (drop (2 ^ m) uinum))

testM :: KnKind -> Int -> Int -> Int -> IC ->
         Int -> IO ()
testM kind m n i bc num = do
    r <- newStdGen
    print ((\(x,y) -> (x/fromIntegral num, y/fromIntegral num))
          (testWithUi kind m n i bc num (take (2^m * num)
          (randoms r :: [Float]))))
```

We now do the same but then with a random Hamming utility instead of just a random utility.

```
testWithUiWH :: KnKind -> Int -> Int -> Int -> IC -> Int ->
                [Float] -> [Bool] -> (Float,Float)
testWithUiWH _ _ _ _ _ 0 _ _ = (0,0)
testWithUiWH kind m n i bc num weights ballots =
  let
  w j = weights !! j
  tb = take m ballots
  ui = uHW w tb m
  meas | kind == Full = manipMeas
       | otherwise = manipMeas0k
  in ((+) (meas m n i (uSet ui) bc priority)
    *** (+) (meas m n i (uSet ui) bc dbRule))
    (testWithUiWH kind m n i bc (num - 1) (drop (2 ^ m) weights)
     (drop m ballots))

testMWH :: KnKind -> Int -> Int -> Int -> IC -> Int -> IO ()
testMWH kind m n i bc num = do
    r <- newStdGen
    print ((\(x,y) -> (x/fromIntegral num, y/fromIntegral num))
```

```
          (testWithUiWH kind m n i bc num (take (2^m * num)
          (randoms r :: [Float])) (take (m * num)
          (randoms r :: [Bool])))
```

Here we calculate the expected win in utility for manipulation. We divide by the average utility.

```
manipMeas0k :: Int -> Int -> Int -> UtilitySet -> IC ->
               AR -> Float
manipMeas0k m n i ui bc rule = let
  tps = truePrefs m (uSingle ui) bc
  bs = models m bc
  avgU = sum (map (uSingle ui) bs) / fromIntegral (length bs)
  kn = [(p1,p2)| p1 <- replicateM i bs, p2 <- replicateM (n-i-1) bs]
  expU b = expectedUBinA  (head (bestBallots bs ui kn m bc rule))
           ui kn m bc rule -
          expectedUBinA b ui kn m bc rule
  maniptot = sum (map expU tps)
  in (maniptot /
      fromIntegral (length tps)) / avgU
```

Get the largest win in utility for several random utilites.

```
testWithUiWC :: KnKind -> Int -> Int -> Int -> IC -> Int ->
                [Float] -> (Float,Float)
testWithUiWC _ _ _ _ _ 0 _ = (0,0)
testWithUiWC kind m n i bc num uinum =
  let
  ui b = uinum !! b2i b
  meas | kind == Full = manipMeas
       | otherwise = manipMeas0k
  old = testWithUi kind m n i bc (num - 1) (drop (2 ^ m) uinum)
  in (maximum [meas m n i (uSet ui) bc priority, fst old],
      maximum [meas m n i (uSet ui) bc dbRule, snd old])

testMWC :: KnKind -> Int -> Int -> Int -> IC -> Int -> IO ()
testMWC kind m n i bc num = do
    r <- newStdGen
    print (testWithUiWC kind m n i bc num (take (2^m * num)
          (randoms r :: [Float])))
```