Actions in Social Choice

MSc Thesis *(Afstudeerscriptie)*

written by

Justin Kruger
(born December 10th, 1987 in Poole, United Kingdom)

under the supervision of **Dr Ulle Endriss**, 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:**
August 29, 2014	Prof Dr Krzysztof Apt
	Prof Dr Jan van Eijk
	Prof Dr Benedikt Löwe (chair)
	Prof Dr Robert van Rooij
	Dr Ulle Endriss

Institute for Logic, Language and Computation

Abstract

This thesis concerns a specific procedure for social choice of a group action under uncertainty. It consists of three main parts. First we investigate formal desiderata for the single agent case. Next the model's position relative to varying disciplines is considered. Finally we provide an implementation of the full multiagent model in the programming language Haskell.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis investigates a specific procedure for aggregating preferences over actions. In this model multiple agents, with differing beliefs and preferences, must decide upon a single joint action to perform. The model itself was introduced by Endriss (2013). It may been seen as coming from two sources: *social choice theory* and *decision theory under uncertainty*. Both these have long, somewhat intertwined histories. For general introductions to the former consult Gaertner (2009) or Arrow et al. (2002). Concerning the latter: the type of decision theory we are dealing with here has roots in work by Wald (1939) and Savage (1972). However, a central characteristic of our approach is *ordinality*, thus the work here is closer to the qualitative version of Dubois et al. (2003).

1.1 Motivation

Suppose there are a group of *agents* who must decide upon a joint *action* to perform. Each agent has potentially different preferences and different beliefs. What aspects should come into play in their decision making process?

This problem is encountered in a variety of domains. For instance, when devising mechanisms for teams of autonomous software agents. In many such cases we desire a *formal procedure* that deterministically determines the best and worst actions to perform. In this thesis we consider one family of such formal procedures. As such we focus on one particular representation of the beliefs and preferences of each agent.

We here model preferences as *weak orders over states* and beliefs as *strict uncertainty sets*, and consider deterministic methods that take these and return a *weak order over actions*. Preferences and beliefs would more typically be represented as utility functions and probability distributions. Our choices are *ordinal* in comparison with these *cardinal* measures. The precise details of our model are given in Section 1.3.

There are a number of reasons why an ordinal framework is interesting. It may not be reasonable to expect all agents to provide a fully specified utility function or probability distribution in all cases. There may also be doubts about utilities and probabilities from a theoretical standpoint. For example, the intra- and interpersonal comparisons afforded by explicit utilities may been seen as unreasonably strong. There may be similar concerns about the interpersonal

comparisons of probability distributions. Once it is decided that either preferences or beliefs should be represented ordinally, it is natural to treat the other in a similar, compatible, manner. Finally, there is a simplicity and elegance to a purely ordinal account, which may even translate practically into computational efficiency.[1]

Our particular representation choices are hopefully justified throughout body of the thesis. In particular, the use weak orders instead of linear orders is a topic of Section 2.4.2. Though strict uncertainty is a very simple model, it is easily extensible as we will see in Section 3.2.

We finish this subsection with some discussion about how uncertainty is introduced into the model. This is achieved, in combination with uncertainty sets, with a *transition function*: that is, a complete description of all the possible outputs of each action. Such transition functions take the elevated position of arguments in the procedure. Hopefully it will become clear how this works in the course of the thesis; for now note that this means we only consider a (potentially small) fixed set of actions at once. This is potentially very different from traditional decision theory in the vein of Savage (1972). However, we tend to compare actions in a pairwise manner regardless of what other actions are possible, in line with consistent comparison of two arbitrary actions. Depending upon your palate for symbols, the transition function is either notational burden or notational clarification.

1.2 Notation and preliminary definitions

From some tastes our use of syntax may be slightly loose. Thus typical first-order logical operators are used (anywhere) with their customary (semantic) meanings: $\neg, \to, \wedge, \forall, \ldots$ Functions are written as $f\colon X \to Y$, or with superscripts as $f\colon Y^X$. Their types are right associative: $f\colon X \to Y \to Z$ is $f\colon X \to (Y \to Z)$. The equality symbol always refers to identity, however the level of this identity may be determined by context. Often juxtaposition is used for function application, in a left associative manner: thus $fx = f(x)$ and $\alpha gy = (\alpha(g))(y)$. Set subscripts typically refer to vectors: $X_Y = (X_y)_{y \in Y}$. Vectors are also be indicated with overlines: $\overline{x}$. Functions may be applied to sets of arguments: for $f\colon X \to Y$ and $A \subseteq X$, let $fA = \{fx \mid x \in A\}$. Similarly, for a collection $F = f_I$ of functions $f_i\colon X \to Y$, given $x \in X$ we have $Fx = \{f_i x \mid i \in I\}$. The inverse of f is f^{-1}. Sets, and some other objects, are typically referred to by capital italics: $A, B, C, \ldots$ Elements and counters are typically referred to by lowercase italics: $a, b, c, \ldots$ Functions tend to be written sans-serif: $\mathsf{F}, \mathsf{s}, \ldots$ Greek letters refer to actions: $\alpha, \beta, \gamma, \ldots$ Calligraphic and bold fonts are also employed, respectively for families of sets and specific sets: $\mathcal{S}, \mathcal{Q}, \ldots, \mathbf{S}, \ldots$ Blackboard fonts are reserved for their common mathematical usage: $\mathbb{N}$ for the natural numbers,...

Preliminary definitions

It is necessary to give some basic definitions concerning binary relations, as there is sometimes inconsistency in terminology. This is especially true across the different disciplines from which we draw.

[1] Though I make no grand claims about the speed of the implementation in Chapter 4.

Definition 1.2.1 (Weak orders)**.** *A binary relation $R \subseteq X \times X$ is* reflexive *if for all $x \in X$, xRx;* antisymmetric *if for all $x, y \in X$, xRy and yRx implies $x = y$;* transitive *if for all $x, y, z \in X$, xRy and yRz implies xRz; and* total *if for all $x, y \in X$, either xRy or yRx. A binary relation that is transitive and total is a* weak order*. Note a weak order is also reflexive. A weak order that is antisymmetric is further a* linear order*. The set of all weak orders over X is $\mathsf{wor}(X)$; of all linear orders is $\mathsf{lin}(X)$.*

Definition 1.2.2 (Weak orders)**.** *The* inverse *R^{-1} of a binary relation R is such that $xR^{-1}y$ iff yRx. For binary relations $R, \trianglerighteq, \succ, \ldots$ we will typically use 'mirror images' to represent inverses: $\bigtriangledown, \trianglelefteq, \prec, \ldots$. The* complement *$R^c$ of a binary relation R is such that xR^cy iff $\neg xRy$. The* strict component *P of a weak order R is the inverse of its complement. This will be called a* strict order*.*

Proposition 1.2.3. *There is a one-to-one correspondence between weak orders and strict orders.*

Definition 1.2.4 (Equivalence classes)**.** *A binary relation I is* symmetric *if xIy implies yIx. The* indifference component *I of a weak order R is the largest symmetric subset of R. This is also an* equivalence relation*: a reflexive, symmetric and transitive relation. For such an equivalence relation $I \subseteq X \times X$, the* equivalence sets *are defined by $\langle \cdot \rangle_I : X \to 2^X$ such that $\langle x \rangle_I = \{y \mid xIy\}$. The* quotient set *is $X/I = \{\langle x \rangle \mid x \in X\}$.*

Definition 1.2.5 (Maximums and minimums)**.** *Let R be a weak order over X. For $Y \subseteq X$, let $\max_R(Y) = \{x \in Y \mid \forall y \in Y,\ xRy\}$; $\max_R = \max_R(X)$. Similarly, let $\min_R(Y) = \{x \in Y \mid \forall y \in Y,\ yRx\}$, $\min_R = \min_R(X)$. We will typically write* max *and* min *and let context determine R. This includes the standard orderings over numbers.*

1.3 The model

We now define the specifics of the model.

Definition 1.3.1 (The model)**.** *We have three primitive sets:*

(i) individuals $i, j, \cdots \in J$,

(ii) actions $\alpha, \beta, \cdots \in A$, *and*

(iii) states $a, b, p, q, \cdots \in Q$.

The belief set *of an individual $i \in J$ is a nonempty set of states:*

$$Q_i \in 2^Q \backslash \{\emptyset\}$$

The preferences (over states) *of an individual $i \in J$ is a weak order over the states:*

$$\trianglerighteq_i \in \mathsf{wor}(Q)$$

For the strict component of state preferences $\triangleright_i$ is used. For the indifference relation $\triangleq_i$ is used.
A profile over beliefs *assigns one belief set to each individual in J; i.e. a vector:*

$$(Q_i)_{i \in J}$$

Similarly, a profile over preferences *assigns preferences to each individual; i.e. a vector* $(\trianglerighteq_i)_{i\in J}$. *Profiles are similarly definable over any structure type.*
A transition function *assigns each action to a sub-function between the states; it is some:*

$$\Delta \in A \to Q \to Q \qquad \textit{(note right associativity)}$$

A transition function first takes an action as argument, then an input *state. It returns an* output *state. A* setup *is a tuple* $(\Delta, \trianglerighteq_J, Q_J)$. *The* full space of setups *is* $\mathbf{S} := (Q^Q)^A \times (\mathsf{wor}(Q))^J \times (2^Q \backslash \{\emptyset\})^J$. *For a subset of possible setups, write* $S \subseteq \mathbf{S}$.
A rule *takes a setup and returns a weak order over actions, i.e. it is a function:*

$$\mathsf{F} : (Q \to Q)^A \times \mathsf{wor}(Q)^J \times (2^Q \backslash \{\emptyset\})^J \longrightarrow \mathsf{wor}(A)$$

For a single agent rule *omit the "J"s above. Images of this function are termed* outcomes*, written*

$$\succcurlyeq \in \mathsf{wor}(A)$$

Such action preferences *have strict component* $\succ$*, and indifference relation* $\simeq$.

Example 1.3.2 (A driving setup)**.** *Imogen and John are on a road trip together. Unfortunately they have gotten lost. The road they are currently taking leads them either to the beach or to the forest. John wants to go to the former and Imogen to the latter. Imogen trusts her map reading skills and believes that they are on the road to the beach. John stopped the car to ask for directions and believes the report that they are heading to the forest. They see a turn-off, which both believe must lead to the other destination: John thinks the beach, and Imogen thinks the forest.*

This may be modelled with primitive sets:

$$Q = \{p_{beach}, q_{forest}\} \qquad A = \{\alpha_{stay}, \beta_{change}\} \qquad J = \{i_{Imogen}, j_{John}\}$$

We drop the text subscripts now and give the belief sets and preferences for each agent: for Imogen we have $Q_i = \{p\}$ *and* $q \rhd_i p$*, for John we have* $Q_j = \{q\}$ *and* $p \rhd_j q$*; and of a transition function* Δ *defined (note juxtaposative function application) by:*

$$\Delta\alpha p = p \qquad \Delta\alpha q = q$$
$$\Delta\beta p = q \qquad \Delta\beta q = p$$

The full setup is then $\big(\Delta, (Q_i, Q_j), (\trianglerighteq_i, \trianglerighteq_j)\big)$.

Note that in Example 1.3.2 both agents 'should' individually come to the same conclusion about the best action, despite having different beliefs and preferences.

1.4 Overview of the thesis

The following chapter, Chapter 2, solely concerns the single-agent case, and will be mostly formal in nature. This is followed by the less formal Chapter 3, in which will discuss some relevant literature and attempt to place this particular approach within. In Chapter 4 we initiate an analysis of the multiagent case, implementing the model in Haskell. We then conclude the main body of the thesis, summing up our results and giving suggestions for future directions. Extra material that does not fit into the main flow is placed in one of two appendices for the single and multiagent cases respectively.

Chapter 2

Single agent case

2.1 Introduction

In considering the single agent case we ignore the set of individuals I. Thus a single agent rule refers to a function defined for some fixed Q and A. A *family of single agent rules* refers to some description of multiple rules for varying Q and A. Unless otherwise stated, for the rest of this chapter we will implicitly be considering an arbitrary single agent rule F, for some fixed Q and A. The symbols Δ, $\trianglerighteq$, and Q_0 will range respectively over transition functions (with action and state arguments), weak orders (over the states), and belief sets (over the states). Assume similar treatment of primed symbols, $\Delta', \trianglerighteq', Q_0', \Delta'', \ldots$ We will make liberal use of these symbols to 'define' rules and axioms in a concise manner. Of course, when doing so care must be taken to ensure such rules are well-defined—as demonstrated by Example 2.3.9 below.

2.1.1 An Example of simple risk

The following example offers an agent a simple gamble.

Example 2.1.1 (Simple risk)**.** *John is enjoying a walk. However, he is in a climate where there is always the possibility of rain. He must decide whether to continue his walk, or to head for shelter. He would prefer to be in the sun if possible, but he definitely doesn't want to get wet.*

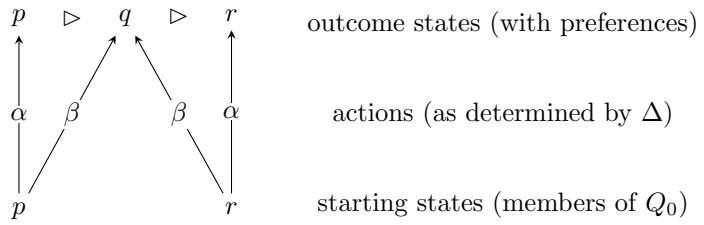
We can model John's choice as follows: $Q = \{p_{\textit{outside-sun}}, q_{\textit{inside}}, r_{\textit{outside-rain}}\}$, $A = \{\alpha_{\textit{risk}}, \beta_{\textit{safe}}\}$. *For the setup, John has beliefs* $Q_0 = \{p, r\}$ *and preferences* $p \rhd q \rhd r$. *Formally, we extrapolate outcomes for the state* q *to define the transition function* Δ*:*

$$\begin{array}{ll} \Delta\alpha p = p & \Delta\beta p = q \\ \Delta\alpha q = q & \Delta\beta q = q \\ \Delta\alpha r = r & \Delta\beta r = q \end{array}$$

The diagram of Figure 2.1 presents all necessary information clearly and concisely.

Roughly, we may classify rules that advise performing β as 'pessimistic', while rules that advise α are 'optimistic'. We now define a single agent rule that takes a middle route.

Figure 2.1 A setup diagram showing a case of simple risk



Example 2.1.2 ('Indifferentistic' rule)**.** *We can assign a score to each state using the preference ranking. Then for each action, sum the scores for each possible output state—this part of the procedure uses the belief set and transition function. Finally, rank the actions by the size of their summed scores. The following formal method is a generalisation of Borda scoring (Fishburn and Gehrlein, 1976).*

Definition 2.1.3 (Simple count scoring)**.** *For a strict order $P \subset X \times X$, the* simple count scoring *is a function $\mathsf{s}_P \colon X \to \mathbb{N}$ with:*

$$\mathsf{s}_P(x) = |\, \{y \in X \mid xPy\} \,|$$

It may be necessary to clarify what set y is intended to range over, for which we add an extra argument. In such cases we write, for $\mathsf{s}\colon X \times 2^X \to \mathbb{N}$:

$$\mathsf{s}_P(x, Y) = |\, \{y \in Y \mid xPy \,|$$

The simple count function assigns increasing values to the different 'levels' of $\rhd$. Note we will also use the corresponding $\trianglerighteq$ here. As we have one-one correspondence between strict orders and weak orders, we write s_R for the same function. Using this we can formally define a rule.

Rule 2.1.4 (Simple count rule)**.** *We define a rule $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ by:*

$$\alpha \succ \beta \quad \textit{iff} \quad \sum_{q \in Q_0} \mathsf{s}_{\trianglerighteq}(\Delta \alpha q) > \sum_{q \in Q_0} \mathsf{s}_{\trianglerighteq}(\Delta \beta q)$$

Applying the rule of Rule 2.1.4 to the setup of Example 2.1.1, each action ends up with 2 points. Thus this rule pronounces indifference in cases of simple risk.

2.1.2 Outline of the chapter

Next, Section 2.2 will look at the shape of single agent rules in more detail, drawing some immediate consequences from our choice of model. After that we look at importing traditional social choice theoretic axioms in Section 2.3, holding off treating the particular family of symmetry axioms (and some extensions) until Section 2.4. Our focus then turns towards investigating rules: output-based in Section 2.5 and casewise-based in Section 2.6. We provide some characterisation results and discuss the compatibility of the two approaches. Section 2.7 concludes.

2.2 Axioms I: the shape of the rule

The shape of our model immediately suggests various consequences and considerations. In this section we explore some of these. Formal desiderata of the procedure are termed *axioms*.

2.2.1 Unrestricted domain

As defined, single agent rules apply to all possible setups; to any member of $\mathbf{S}$. The size of this space directly depends on how many different instances of each of the three coordinates are possible. Thus, multiplying the following three values gives $|\mathbf{S}|$.

1. The number of possible belief sets is straightforward: $2^{|Q|} - 1$.
2. The number of possible preferences is also solely dependent upon the cardinality of Q, but it does not have so simple a formulation. For a set with n elements, the number of possible weak orders is the nth ordered bell number—sequence A000670 in the On-Line Encyclopedia of Integers (OEIS). 'Small' examples are given in Table 2.1.

Table 2.1: Number of distinct weak orders over sets of small cardinalities.

Set size	2	3	4	5	6	7	8	9
Number of weak orders	3	13	75	541	4683	47293	545835	7087261

3. There are $|Q|^{|Q||A|}$ possible transition functions. Note this partially depends upon $|A|$; we are *not* counting the number of *possible actions*.

The requirement that all possible inputs to a function are defined is often termed "universality" or "unrestricted domain". Though in one sense this is implicit in the model as defined, it may be violated in spirit by modifying the scope of subsequent axioms. That is, we may want the definitions and axioms that follow *not* to apply across all possible setups, but only for some subset of them.

Definition 2.2.1 (Subspace)**.** *A* subspace *of the set of all setups is simply a subset* $S \subseteq \mathbf{S}$.

In specific cases such restrictions will be necessary (e.g. Definition 2.3.3 below).

2.2.2 Trivial cases

One consequence of having unrestricted domains is that there are some seemingly trivial cases, where the output of a single agent rule seems obvious.

Axiom 2.2.2 (Single belief state determines action ranking)**.** *If only one input state is thought possible, then the action ranking should mimic the ranking over the states the actions go to from this input state. Formally, for* $\mathsf{F}(\Delta, \trianglerighteq, \{q\}) = \succcurlyeq$*, require:*

$$\alpha \succcurlyeq \beta \quad \textit{iff} \quad \Delta\alpha q \trianglerighteq \Delta\beta q$$

Note that alongside the implicit unrestricted domain this implies a version of *non-imposition* (see Appendix A.1). There are similar trivial cases involving the preferences over states and the transition function.

Axiom 2.2.3 (Indifferent states implies indifferent actions)**.** *The trivial preference over states results in indifference between all actions. Formally and conversely, for* $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$*:*

$$\exists\, \alpha, \beta \in A,\ \alpha \succ \beta \quad \rightarrow \quad \exists\, a, b \in Q,\ a \rhd b$$

Axiom 2.2.4 (Indifference of indistinguishable actions)**.** *If two actions always end up in indifferent states, the rule should be indifferent between the two. Given* $\alpha, \beta \in A$*, for an single agent rule with* $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$*:*

$$(\forall q \in Q_0\ ,\ \Delta\alpha q \triangleq \Delta\beta q) \quad \rightarrow \quad \alpha \simeq \beta$$

2.2.3 Irrelevance conditions

Under our initial interpretation, those states not in the uncertainty set are states that are thought *impossible*, and should thus *have no affect on the outcome.*[2] Here this is formalised in terms of the transition function.

Axiom 2.2.5 (Irrelevance of impossible states)**.** *If two transition functions act the same on states that are considered possible, the outcome should be the same. If for all actions* α *and all states* $p \in Q$*, if* $\Delta\alpha p = \Delta'\alpha p$ *then:*

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q_0)$$

Axiom 2.2.5 is implicit in our drawing of setup diagrams, which do not even show the output of actions from impossible states. We give axioms of this type the general name *irrelevance axioms*. These proceed by delineating an 'area' of setups which, when changed, does not affect the outcome.

Axiom 2.2.6 (Irrelevance of unreachable states)**.** *Preferences over states that are not reachable from any believed state by any action are irrelevant to the outcome. Formally, consider* $\Delta A Q_0$*—the set of possibly reachable states. If* $\trianglerighteq$ *and* $\trianglerighteq'$ *are such that for all* $p, q \in \Delta A Q_0$*,* $p \trianglerighteq q$ *iff* $p \trianglerighteq' q$*, require:*

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta, \trianglerighteq', Q_0)$$

2.3 Axioms II: treating states-as-voters

One possible 'metaphor' here is to treat *states-as-voters*. Each state in the belief set can be thought of as a *voter*, whilst the actions become *candidates*. This enables a partial return to the social choice framework.

Definition 2.3.1. *For a given setup, for each* $q \in Q$*, the* induced preference (over actions) *is a binary relation* $\succcurlyeq_q \in \mathsf{wor}(A)$ *with:*

$$\alpha \succcurlyeq_q \beta \quad \textit{iff} \quad \Delta\alpha q \ \trianglerighteq\ \Delta\beta q$$

[2] We will consider a more nuanced view in Section 3.2.

Different transition function and state preference combinations may induce the same preferences over actions. We may want such setups to have the same outcomes over F: see Section 2.6 and Axiom 2.6.4. If we also restrict attention to setups with some fixed set of possible states, we complete the return to the social choice theoretic framework.

Definition 2.3.2 (Voter subspace)**.** *A* voter subspace*, defined in terms of a fixed belief set Q_0 is a subset of all possible setups:*

$$S_{Q_0} = \{(\Delta, \trianglerighteq, Q_0) \mid \Delta \in (Q^Q)^A, \trianglerighteq \in \mathsf{wor}(Q)\}$$

That is, if we consider the induced preference on a voter subspace, we effectively consider all possible rankings by a set of voters, and how these can be aggregated by some social aggregation procedure. Note the set of all voter subspaces partitions the full space. The immediate question is how traditional axioms can be defined *across* different voter subspaces.

In the rest of this section we look at various ways of importing social choice theoretic axioms. This includes the classic axioms of Arrow (1963): Non-dictatorship, Pareto, and Independence of irrelevant alternatives; as well as various others.[3]

2.3.1 Favoured state

Treating states-as-voters, a 'dictator' becomes a state that is strongly favoured over the others in some way.

Definition 2.3.3 (Favoured state)**.** *A state q is (strongly) favoured in some subspace S if, for setup arguments $(\Delta, \trianglerighteq, Q_0) \in S$, its strict(/weak) induced preferences over actions are copied in the outcome.*

$$\Delta\alpha q \rhd \Delta\beta q \quad \rightarrow \quad \alpha \succ \beta$$
(for strong, substitute $\trianglerighteq$ for $\rhd$, and $\succcurlyeq$ for $\succ$)

Under Axiom 2.2.5, any favoured state must be a member of Q_0. Thus a state will typically only be favoured relative to a subset of a voter subspace. A non-dictatorship condition ensures that some given subspaces of all setups do *not* have a favoured state.

Axiom 2.3.4 (No favoured states)**.** *Let $\mathcal{S}$ be a collection of subsets of the full space of setups. There are* no favoured states over $\mathcal{S}$ *if no state is favoured for any element $S \in \mathcal{S}$.*

For a true parallel to non-dictatorship, we apply this to all non-trivial voter subspaces, where trivial voter subspaces are those where Q_0 is a singleton. Such cases clearly correspond to the trivial case of a single voter; note also by Axiom 2.2.2 such *must* have favoured states.

Axiom 2.3.5 ("No-dictator" analogue)**.** *Let $\mathcal{S} = \{S_{Q_0} \mid |Q_0| > 1\}$. The* "no-dictator" analogue *requires no favoured states over $\mathcal{S}$.*

[3] Some of which are relegated to the Appendix. Richelson (1977) provides a large list of axioms concerning social choice functions.

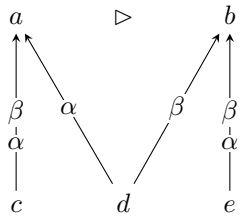
Proposition 2.3.6. *If a subspace S has a favoured state then this is also favoured in all $S' \subseteq S$. Similarly, if there is no favoured state over two families of subspaces $\mathcal{S}$ and $\mathcal{S}'$, then there is no favoured state over $\mathcal{S} \cup \mathcal{S}'$.*

Proof. Straightforward. □

2.3.2 Casewise dominance

'Pareto' dominance occurs when one option is better along all 'dimensions' than another. Here, dimensions are possible states, and the relative merit of options—that is, actions—is determined by the ranking of the output states. For example, in cases like Figure 2.2, one action is clearly 'better' than another.

Figure 2.2 Setup diagram where one action is at least as good as another in all possible states.



Consider the model to the left. It seems obvious that α is the best possible action: for each possible input state doing α results in an state that is at least as good as, and in some cases better than, doing β.

The following is equivalent to **Definition 7** by Endriss (2013).

Definition 2.3.7 (Casewise dominance[4])**.** *Given a setup, an action α* casewise dominates *β if: if for every state $q \in Q_0$ we have $\Delta\alpha q \trianglerighteq \Delta\beta q$, and for some $p \in Q_0$, $\Delta\alpha p \rhd \Delta\beta p$, then $\alpha \succ \beta$.*

Axiom 2.3.8 (Casewise dominance)**.** *If one action casewise dominates another, it should be strictly preferred.*

Though this is obviously a desirable condition, it cannot be deployed on its own to define a rule.

Example 2.3.9 (Intransitivity paradox)**.** *Suppose we attempt to define a rule $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ by $\alpha \succ \beta$ iff α casewise dominates β. It is easy, as in Figure 2.3, to create a setup with three actions but only one case of casewise dominance. By casewise dominance and transitivity of indifference it follows that one action should both be preferred to and indifferent to another: thus the rule is not well-defined.*

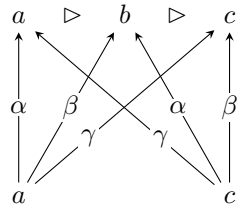
It is also fairly obvious that attempting to rank pairs of actions by counting how many input states one performs better than another will lead to Condorcet style problems. Regardless of these issues, casewise dominance is hence considered a minimal requirement for an attractive rule.

Proposition 2.3.10. *Casewise dominance is inherited by subspaces, and is preserved under the union of subspaces.*

Proof Sketch. Casewise dominance concerns itself with one setup at a time. □

[4]It might be most accurate to term this *strong* casewise dominance, as we do not require dominance in *every* case/state. But that extra "strong" is needless, as we do not consider 'true' casewise dominance.

Figure 2.3 Setup with three actions and a single case of casewise dominance, demonstrating the intransitivity of casewise *non*-dominance.



Here α casewise dominates β. However, there are no other cases of casewise dominance: β performs better than γ from a and γ performs better than both α and β from c.

2.3.3 Independence

Independence conditions may be viewed as a generalisation of the above irrelevance axioms. For independence conditions, *two* 'areas' are delineated. Changes to area (i) are required to be irrelevant to the outcome *restricted to* area (ii). We then say that the area (ii) is *independent* from area (i). For example, we may feel that the relative ranking of two actions is independent of what the transition function does to *other* actions. This may be phrased contrapositively as: the outcome restricted to area (ii) *only depends upon* (the complement of) area (i).

Axiom 2.3.11 (Weak pairwise transition independence)**.** *The relative ranking of two actions only depends upon what these two actions output. Take a single agent rule with* $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ *and* $\mathsf{F}(\Delta', \trianglerighteq, Q_0) = \succcurlyeq'$*. For any pair of actions* α, β*; if* $\forall q \in Q_0$*,* $\Delta\alpha q = \Delta'\alpha q$ *and* $\Delta\beta q = \Delta'\beta q$ *then* $\alpha \succcurlyeq \beta \leftrightarrow \alpha \succcurlyeq' \beta$*.*

The choice of a pairwise—or *binary*—comparison here mirrors that of the Arrovian *independence of irrelevant alternatives* (hereafter *Arrovian independence*).[5] However, in contrast to Arrow's 1963 theorem, many rules satisfy Axioms 2.3.5, 2.3.8 and 2.3.11—i.e. non-dictatorship, casewise dominance and weak-pairwise-transition independence. Indeed, the simple count of Rule 2.1.4 suffices.

This is because the Arrovian *independence of irrelevant alternatives* is 'stronger' than weak-pairwise-transition independence. Weak pairwise independence corresponds closer to independence of other alternatives *if also* the positions of both alternatives are fixed. To give an example in the Arrovian framework, if candidate α is ranked fifth and candidate β is ranked third by the first voter, and second and third by the second voter, etc.; if we keep these ordinal *positions* for each voter, then even if we change the other candidates positions the outcome ranking between α and β should remain the same. For a closer analogue to Arrovian independence, we want something more like:

> the relative action ranking between two actions only depends upon the relative ranking of the *output states* of these actions.

Axiom 2.3.12 (Strong independence)**.** *The relative ranking of two actions only depends upon the ranking between their outputs for each input. For* $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ *and* $\mathsf{F}(\Delta', \trianglerighteq', Q_0) = \succcurlyeq'$

[5] Further, extensions to the ternary, quaternary, etc. cases are shown to reduce to the binary case by Blau (1971).

$$\forall q \in Q \; (\; \Delta\alpha q \; \trianglerighteq \; \Delta\beta q \quad \leftrightarrow \quad \Delta'\alpha q \; \trianglerighteq' \; \Delta'\beta q \;) \\ \rightarrow \quad (\; \alpha \succcurlyeq \beta \quad \leftrightarrow \quad \alpha \succcurlyeq' \beta \;)$$

Proposition 2.3.13. *Strong independence is inherited by subspaces of* **S**. *It is* not necessarily *preserved by the union of such.*

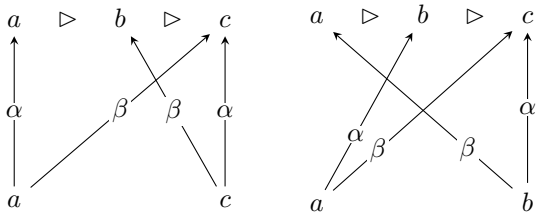
Proof. Strong independence involves the comparison of two setups. Clearly the set of possible pairs over a subset is itself contained in the set of possible pairs over the larger set. Thus if the axiom holds for all the pairs, it will hold for the smaller set of pairs also. Also clearly the converse does not hold: in taking the union of two different sets there are new pairs possible, for which the axiom may fail. □

Proposition 2.3.14 (Arrow)**.** *For $Q > 1$ and $A > 3$, no rule satisfies the non-dictatorship analogue, casewise dominance and strong independence; Axioms 2.3.5, 2.3.8 and 2.3.12.*

Proof Sketch. Take any (non-trivial) voter subspace (possible as $Q > 1$). The result is a direct application of Arrow's theorem, given the inheritance of the conditions by Proposition 2.3.10 and Proposition 2.3.13 to the subspace.[6] □

Though probably the most direct translation of Arrovian independence, strong independence may be thought *too* strong for current purposes. For instance, it implies that the two setups of Figure 2.4 must have the same outcome. What may be wanted instead is something that 'can tell' when there is a 'crossing' between an output of α and an output of β, *regardless of the inputs involved.*

Figure 2.4 Two setups that are indistinguishable under strong independence.



This is a problem if you want to rank $\alpha \succ \beta$ for the left model and $\beta \succ \alpha$ for the right.

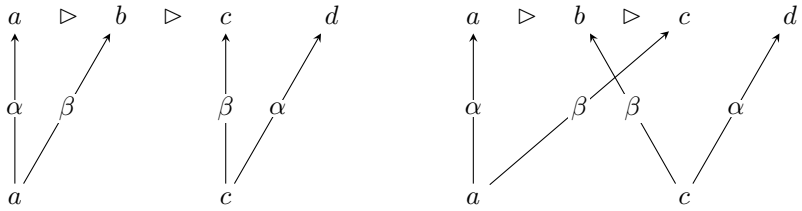
Axiom 2.3.15 (Crossing independence)**.** *The relative ranking of two actions only depends upon the ranking between each output and all the outputs for the other action. Take a single agent rule with $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ and $\mathsf{F}(\Delta', \trianglerighteq', Q_0) = \succcurlyeq'$. Require*

$$\forall p, q \in Q \; (\; \Delta\alpha p \; \trianglerighteq \; \Delta\beta q \quad \leftrightarrow \quad \Delta'\alpha p \; \trianglerighteq' \; \Delta'\beta q \;) \\ \rightarrow \quad (\; \alpha \succcurlyeq \beta \quad \leftrightarrow \quad \alpha \succcurlyeq' \beta \;)$$

[6] Three elegant proofs of Arrow's theorem are given by Geanakoplos (2005).

Note this concerns changes to both the transition function and the preferences over states. It is designed to be a *minimal*[7] weakening of strong independence that allows to distinguish the setups of Figure 2.4. It *cannot* distinguish the crossing of outputs of a single action; the setups of Figure 2.5 must give the same outcome under this axiom.

Figure 2.5 Two indistinguishable setups under crossing independence.



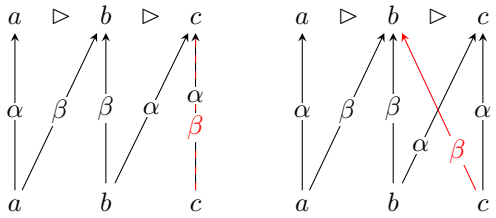
We leave unfinished the question of to what extent the crossing independence is compatible with casewise dominance, instead finishing this subsection with the obvious generalisation of Proposition 2.3.13.

Proposition 2.3.16. *All the independence axioms of this subsection are inherited by subspaces, but not preserved by unions thereof.*

2.3.4 Monotonicity and positive responsiveness

Suppose that two setups are identical *except* one action has one input that goes to a better output, as in Figure 2.6.

Figure 2.6 Two setups involving an improvement to a single action.



Pessimists and optimists might disagree about whether β is the best action to perform in the left model. But *if it is* thought best, then it surely should also be thought best in the right model.

Axiom 2.3.17 (Monotonicity)**.** *Changing the transition function by making one action have a better output should not decrease the position of this action in the outcome. Take a single agent rule with* $f(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ *and* $f(\Delta', \trianglerighteq, Q_0) = \succcurlyeq'$*. Require:*

$$\begin{array}{l} \forall q \in Q_0,\ \Delta'\alpha q \trianglerighteq \Delta\alpha q \\ \qquad\qquad \& \\ \exists! q \in Q_0,\ \Delta'\alpha q \trianglerighteq \Delta\alpha q \\ \qquad\qquad \& \\ \forall \beta \neq \alpha,\ \forall q \in Q_0,\ \Delta'\beta q = \Delta\beta q \end{array} \quad \rightarrow \quad (\quad \alpha \succcurlyeq \beta \quad \rightarrow \quad \alpha \succcurlyeq' \beta \quad)$$

[7] Arguments as to why such a minimal weakening is desirable will be given in Section 3.3.

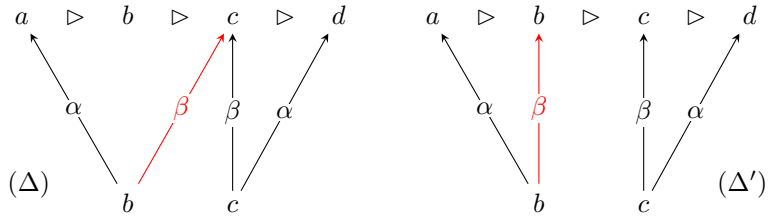
One may also think that improvements should break ties. Thus, if α and β are thought indifferent in the left setup of Figure 2.6, β should be preferred to α in the right setup. This gives a simple strengthening of monotonicity.

Axiom 2.3.18 (Strong positive responsiveness)**.** *As Axiom 2.3.17, and any ties concerning the 'improved' action should be broken in its favour. Formally, require:*

$$\begin{array}{l} \forall q \in Q_0,\ \Delta'\alpha q \trianglerighteq \Delta\alpha q \\ \qquad\qquad \& \\ \exists! q \in Q_0,\ \Delta'\alpha q \rhd \Delta\alpha q \\ \qquad\qquad \& \\ \forall \beta \neq \alpha,\ \forall q \in Q_0,\ \Delta'\beta q = \Delta\beta q \end{array} \quad \rightarrow \quad (\quad \alpha \succcurlyeq \beta \quad \rightarrow \quad \alpha \succ' \beta \quad)$$

However, some improvements to actions may be thought irrelevant, as in Figure 2.7 . We may only want ties to be (forcibly) broken if the improvement 'shifts past' an output of the tied action.

Figure 2.7 An example of a potentially irrelevant improvement to an action.



Definition 2.3.19. *For a state $q \in Q$ and action γ let:*[8]

$$\mathsf{b}(q,\gamma) = |\{p \in Q_0 \mid \Delta\gamma p \rhd q\}| - |\{p \in Q_0 \mid q \rhd \Delta\gamma p\}|$$

In words, b *counts how many outputs of the action γ are better and worse than some state q.*

For example, in Figure 2.6, the left setup has $\mathsf{b}(c,\alpha) = 1$ and $\mathsf{b}(b,\alpha) = -1$. Whereas in Figure 2.7, the setups have $\mathsf{b}(c,\alpha) = \mathsf{b}(b,\alpha) = 0$. The relevant point is that b and c are the outputs states by which the *other* action, β, improves.

Axiom 2.3.20 (Weak positive responsiveness)**.** *As Axiom 2.3.17 (monotonicity), and if the improved output becomes ranked higher than the output of another action, ties with this other action should be broken. Take a single agent rule with $f(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ and $f(\Delta', \trianglerighteq, Q_0) = \succcurlyeq'$. Suppose for all actions γ and states p, $\Delta\gamma p = \Delta'\gamma p$; except that for two actions α, β and a single state $q \in Q_0$, for which $\Delta\alpha q \neq \Delta'\alpha q$ and we have:*

$$\mathsf{b}(\Delta'\alpha q, \beta) < \mathsf{b}(\Delta\alpha q, \beta)$$

Then $\alpha \succcurlyeq \beta$ implies $\alpha \succ' \beta$.

[8] Note this is relative to a transition function and state preference: we should really write $\mathsf{b}_{\Delta,\trianglerighteq}(\cdot)$.

This version arguably chimes better with Axiom 2.3.15 (crossing independence).

Proposition 2.3.21. *Monotonicity type axioms are inherited by subspaces, but not preserved by the union of subspaces.*

Proof. As in the proof of Proposition 2.3.13. □

2.4 Symmetry and beyond

Positive responsiveness is used by May (1952) to characterise the simple majority rule in combination with another axiom: *anonymity*. This is an example of a *symmetry* condition. Such symmetry conditions can usually be expressed in terms of invariance of the outcome given some permutation. They are given distinctive names depending upon what the permutation acts upon: so anonymity for individuals; neutrality for candidates.

2.4.1 Importing symmetry axioms

Continuing the states-as-voters interpretation, our first axiom here corresponds to neutrality.

Axiom 2.4.1 (Action symmetry)**.** *Permuting the actions within a transition function results in a similarly permuted outcome. Let $\sigma\colon A \to A$ be a permutation on the actions. Take two transition functions such that the permutation maps one to the other, i.e. Δ, Δ' such that for all states $q \in Q_0$:*

$$\Delta(\sigma\alpha)q = \Delta'\alpha q$$

We then require, for $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ and $\mathsf{F}(\Delta', \trianglerighteq, Q_0) = \succcurlyeq'$, for all pairs of actions α, β:

$$\alpha \succcurlyeq \beta \quad \leftrightarrow \quad \sigma\alpha \succcurlyeq' \sigma\beta$$

Our next axiom corresponds to anonymity.

Axiom 2.4.2 (Transition-input symmetry)**.** *Permuting believed states in the input of the transition function does not change the outcome. For a given setup, let $\sigma\colon Q_0 \to Q_0$ be a permutation. If for all actions $\alpha \in A$ and states $q \in Q$, $\Delta\alpha(\sigma q) = \Delta'\alpha q$, then:*

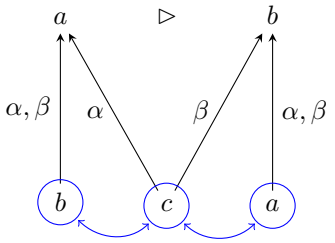
$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q_0)$$

The basic idea of Axiom 2.4.2 is demonstrated by Figure 2.8. The underlying principle is that the label of the input state does not matter. This leads to a natural extension of the anonymity-like axiom to include impossible states in the permutation.

Axiom 2.4.3 (Belief-state symmetry)**.** *Permuting the states within the belief set and in the input to the transition function does not change the outcome. Let $\sigma\colon Q \to Q$ be a permutation. Let $Q'_0 = \sigma Q_0$. Let $\Delta'\alpha q = \Delta\alpha(\sigma q)$ for all actions α and states $q \in Q$. Require:*

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q'_0)$$

Figure 2.8 A visual example of permuting the inputs of a transition function.



Changing around the input states at the bottom should not affect the ranking $\alpha \succ \beta$.

Alongside belief-state symmetry, we consider an extension of a voter subspaces—recall $S_{Q_0} = \{(\Delta, \trianglerighteq, Q_0) \mid \text{arb. } \Delta, \trianglerighteq\}$.

Definition 2.4.4 (Anonymous-voter subspace)**.** *An* anonymous-*voter subspace is, for some integer $n \leq |Q|$:*

$$S_n = \bigcup \{S_{Q_0} \mid |Q_0| = n\}$$

In the above Axiom 2.4.3 we extended the *scope* of the permutation in two ways: (first) the permutation itself was extended to range over the full set of states; which allowed the permutation to be (second) non-trivially applied to belief sets as well as the transition function. What about the most general scope, where we apply the fully extended permutation throughout all of the setup?

Axiom 2.4.5 (Global-state symmetry)**.** *Permuting the states in the belief set, preference relation and transition function with a uniform permutation should not change the outcome.*

However, note that creating an axiom by simply permuting all states is by no means obviously sensible. Indeed, permuting the states in a preference relation *would likely change the outcome* for any reasonable rule. In the neutrality of traditional voting theory, permuting the candidates of the preferences is only sensible alongside a parallel permutation of the outcome. In Axiom 2.4.1, this is manifested as a permutation of both the transition function and the outcome. In fact, Axiom 2.4.5 *is* reasonable, as it similarly contains two 'cancelling' permutations: the preference permutation and the permutation of the *output* states of the transition function.

We identify three underlying, basic, and somewhat reasonable targets for permutations of states here. The first we have seen in Axiom 2.4.3: permute states in the belief set. The second we have intimated in the above paragraph: permute states in both the preference relation and the *output* of the transition function.

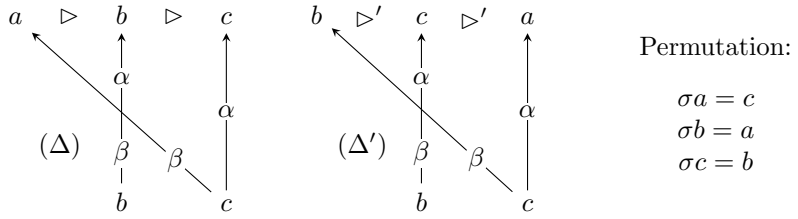
Axiom 2.4.6 (Output-preference symmetry)**.** *Permuting states in the state preference and output states does not change the outcome. Let $\sigma \colon Q \to Q$ be any permutation. If for all pairs of states p, q and actions α we have:*

$$\sigma p \trianglerighteq \sigma q \iff p \trianglerighteq' q \quad \text{and} \quad \sigma(\Delta \alpha p) = \Delta' \alpha p$$

then we require:

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta' \trianglerighteq', Q_0)$$

Figure 2.9 Two setups with permuted states in both preferences and output of transition function.



See Figure 2.9 for a simple example.

The final basic symmetry is to permute the states of the *input* of an *individual action*.

Axiom 2.4.7 (Action-input symmetry)**.** *For a given action, permuting its inputs does not change the outcome. Formally, for each action take a permutation* $\sigma_\alpha \colon Q_0 \to Q_0$*. Take* Δ *and* Δ' *such that for all actions* $\alpha \in A$ *and for all states* $q \in Q_0$*:*

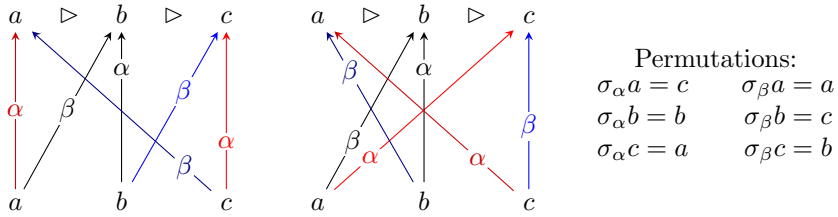
$$\Delta\alpha(\sigma_\alpha q) = \Delta'\alpha q$$

Require:

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q_0)$$

According to Axiom 2.4.7 the setups in Figure 2.10 would have the same outcomes.

Figure 2.10 Two setups with permuted inputs for individual actions.



At the most general level we could have a different permutation for each of these basic levels: that gives $|A| + 2$ possible axes of symmetry for states alone (recall that for actions we also have Axiom 2.4.1). However, we can also combine these to get weaker conditions. For example, in Axiom 2.4.7 we may require that all the action-inputs undergo the same permutation; which basically amounts to transition-input symmetry, Axiom 2.4.2. Note that Axioms 2.4.3, 2.4.6, and (2.4.7 or 2.4.2) together imply Axiom 2.4.5, but that no implication holds (simpliciter) in the other direction.

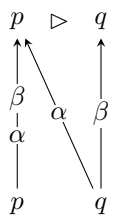
2.4.2 States vs. possible outputs

The rest of this section works towards justifying our choice of weak orders rather than linear orders. The rest of this section works towards justifying this

statement, with reference to a generalisation of the symmetry axioms.

The basic underlying rationale is that there is no restriction on the output states of actions. One action can have the same output from different input states, and two actions can have the same output from the same input, as demonstrated in Figure 2.11. The *states* (over which preferences are expressed),

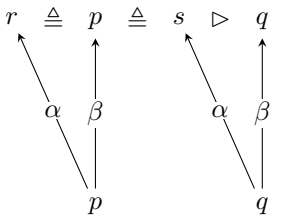
Figure 2.11 Examples of 'indifferences'.



The following applies to both the left and right models:

$$\Delta\alpha p \triangleq \Delta\beta p \triangleq \Delta\alpha q$$

We must allow transition functions as in the left setup; so why not allow state preferences as in the right setup?



need to be separated from the *possible outputs* (those states mapped to by some action from some possible state). For (i) not every state need be a possible output and (ii) two distinct possible outputs may be the same state. By (ii) we have cases of effective indifference between possible outputs. Arguably we are only concerned with the preferences over the possible outputs—cf. Axiom 2.2.6 (irrelevance of unreachable states). But the symmetry axioms defined above concern permutations of *all* the states. Roughly speaking: symmetry properties concern *automorphisms* between structures; but we can preserve the structure of preferences over possible outputs with less demanding transformations. Continuing the slightly rough language, we are here concerned with *isomorphisms* between sets of possible outputs, or *(strong) homomorphisms* between sets of states.

Reducing weak orders to linear orders

The idea is partially demonstrable by 'reduction' of setups involving weak orders to ones solely involving linear orders. Or, in the opposite direction, we extend a rule defined on linear preferences to be defined upon preferences over weak orders. The procedure in short: take equivalence classes of states with respect to the weak order; modify the setup accordingly, and apply the relevant linear single agent rule.

Recall that an unrestricted domain is built into the framework: single agent rules are automatically defined for all weak orders. However, we can restrict attention to linear orders.

Definition 2.4.8 (Linear ordering subspace)**.** *The* linear order subspace *is the subspace defined by:*

$$S_{\mathsf{lin}} = \{\ (\Delta, \trianglerighteq, Q_0)\ \mid\ \trianglerighteq \in \mathsf{lin}(Q)\,,\quad \textit{arbitrary}\ \Delta, Q_0\}$$

Definition 2.4.9 (Extending rules on Linear orderings)**.** *Suppose we have a 'rule'* G *defined on (or satisfying axioms restricted to)* S_{lin}*. We extend this*

to a full single agent rule F *defined on the full space of setups. Take an arbitrary setup* $(\Delta, \trianglerighteq, Q_0)$*. We consider the quotient set* $Q/\triangleq$ *(recall Definition 1.2.4). To this we add a set of (new) dummy states* $\{p_1, p_2, \dots\} = P$ *such that* $\left|P \cup Q/\triangleq\right| = |Q|$*. Thus we have a bijection* $h\colon P \cup Q/\triangleq \to Q$ *(any will do). On* $P \cup Q/\triangleq$ *we define a linear order* $\trianglerighteq'$ *by:*

$$\begin{array}{lll} \langle x\rangle & \trianglerighteq' & \langle y\rangle \qquad \textit{iff} \quad x \trianglerighteq y \\ \langle x\rangle & \triangleright' & p_i \qquad \textit{for all } x \in Q \textit{ and for all } p_i \in P, \\ p_i & \trianglerighteq' & p_j \qquad \textit{iff} \quad i \geq j \end{array}$$

The transition function Δ' *is defined for actions* $\alpha \in A$ *and states* $q \in P \cup Q/\triangleq$ *as:*

$$\Delta'\alpha q = \langle \Delta\alpha(hq)\rangle$$

The belief set becomes $Q'_0 = h^{-1}Q_0$*. The single agent rule* F *is then simply defined as:*

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = G(\Delta', \trianglerighteq', Q'_0)$$

Strengthening symmetry axioms

This reduction is captured by a strengthening of Axiom 2.4.6 (output-preference symmetry) to the case of (strong) homomorphisms rather than that of automorphisms, in combination with the other symmetry axioms of Section 2.4.1. Roughly, if we think of the states as the elements of our model, and the signature as consisting of action-state pairs as terms and $\trianglerighteq$ as a binary relation between these, we have an endomorphism between the states, with respect to the possible outputs and preferences over these. The strengthened axiom is:

Axiom 2.4.10 (Invariance between 'isomorphic' possible outputs)**.** *Changing states in such a manner as to preserve preference relations between possible outputs does not affect the output. Suppose for all actions* $\alpha, \beta \in A$ *and states* $q, r \in Q_0$ *we have:*

$$\Delta\alpha q \ \trianglerighteq\ \Delta\beta r \quad \textit{iff} \quad \Delta'\alpha q \ \trianglerighteq'\ \Delta'\beta r$$

then:

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq', Q_0)$$

This section treated *output states* as a possible item of analysis. In the next section these become the focus for defining rules.

2.5 Output-based approach

Endriss (2013) distinguishes two general approaches to rules. We re-dub the first of these the *output-based* approach.[9] In contrast with the casewise-based approach, the output-based approach restricts attention to what states are achievable by each action. An ad hoc characterisation is given in the Appendix as Axiom A.1.4. We consider the simplest version first.

[9] Changed from "outcome-based" approach. We reserve "outcome" for the image of rules.

2.5.1 Output sets

Definition 2.5.1 (Output sets)**.** *Given a transition function, belief set, and action, the* (possible) output set *is the set of outputs that the action goes to from the belief set, i.e.:* $\Delta\alpha Q_0$.

There is an extensive literature on methods for *ranking sets of objects* (Barbera et al. (2004) provides a survey); methods which take a weak order over elements and return a weak order over *sets* of elements. Any such method defines a rule, whereby the actions $(\alpha, \beta, \dots)$ are ordered according to how their corresponding output sets $(\Delta\alpha Q_0, \Delta\beta Q_0, \dots)$ are ranked according to the set ranking extended from $\trianglerighteq$.

Rule 2.5.2 (Set-based pessimistic)**.** *Compare the least preferred state in each output set. Formally,* $\alpha \succcurlyeq \beta$ *iff* $\min(\Delta\alpha Q_0) \trianglerighteq \min(\Delta\beta Q_0)$

Rule 2.5.3 (Set-based optimistic)**.** *Compare the most preferred state in each output set. Let* $\alpha \succcurlyeq \beta$ *iff* $\max(\Delta\alpha Q_0) \trianglerighteq \max(\Delta\beta Q_0)$.

The above two rules may be extended and combined in various ways. The following is due to Arlegi (2002).

Rule 2.5.4 (Set-based minmax)**.**

$$\begin{array}{lll} \textit{Let} \quad \alpha \succcurlyeq \beta & \textit{iff} & \min(\Delta\alpha Q_0) \triangleright \min(\Delta\beta Q_0) \\ & & \quad \textit{or} \\ & & \min(\Delta\alpha Q_0) = \min(\Delta\beta Q_0) \ \textit{and} \ \max(\Delta\alpha Q_0) \trianglerighteq \max(\Delta\beta Q_0) \end{array}$$

For maxmin, permute min and max in the above.

There is an immediate unsatisfactory property of using rankings of sets of objects: casewise dominance is violated (the output sets of Figure 2.2 provide a demonstration). This problem is readily solvable by moving to consider *multisets*: sets which allow *multiple* occurrences of the same element.[10]

2.5.2 Basic properties of multisets

Multisets are less seen in the wild than ordinary sets, and classifications by hunters vary; so first some taxonomy.[11]

Definition 2.5.5 (Multisets)**.** *A* multiset X *over* $\{1, 2, \dots, n\} = \mathbb{N}_n$ *is:*

$$X = \{\!\{(1, i_1), (2, i_2), \dots, (n, i_n)\}\!\}$$

where i_x *records how elements of type* x *there are. This is also expressible by a multiplicity function:*

$$\mu_X \colon \mathbb{N}_n \to \mathbb{N} \qquad \textit{with} \qquad \mu_X(x) = i_x$$

The 'cardinality' of a multiset is the sum of these multiplicities:

$$|X| = \sum_x \mu_X(x)$$

[10] Endriss (2013) remains in the set-based paradigm and thus concludes that the output-based approach is incompatible with the casewise-based. But cf. Proposition 2.6.8.

[11] For instance, one might want to distinguish between multiset *union* and *addition*, as in for e.g. Girish and Sunil (2009).

'Membership' occurs if the multiplicity is not zero:

$$x \in X \leftrightarrow \mu_X(x) > 0$$

Multiset 'union' $\uplus$ *sums the multiplicities by element:*

$$\mu_{X \uplus Y}(x) = \mu_X(x) + \mu_Y(x)$$

The width *of a multiset* wd *is the cardinality of its set of elements:*

$$\mathsf{wd}(X) = |\{x \mid x \in X\}|$$

The set of all (finite) multisets of $\mathbb{N}_n$ *is denoted by* $\mathcal{N}_n$*. The set of all finite multisets of* $\mathbb{N}$ *is then* $\mathcal{N} = \bigcup_{n \in \mathbb{N}} \mathcal{N}_n$*. Let:*

$$\mathcal{N}_{n,k} = \{X \in \mathcal{N}_n \mid |X| = k\} \qquad \mathcal{N}_{*,k} = \{X \in \mathcal{N} \mid |X| = k\}$$

Proposition 2.5.6. *The number of multisets of cardinality* k *with elements taken from* $\mathbb{N}_n$ *is*

$$|\mathcal{N}_{n,k}| = \binom{n+k-1}{n}$$

Proof Sketch. (Wikipedia) imagine a multiset as n dots split into k partitions. It thus has $k-1$ 'dividers', e.g.:

$$\{\!\{a, a, a, a, b, b, c, c, c, d\}\!\}$$
$$\downarrow$$
$$\bullet\bullet\bullet\bullet \mid \bullet\bullet \mid \bullet\bullet\bullet \mid \bullet$$

The total number of characters (dots and dividers) is $k + n - 1$. Thinking of each position as a distinct object, the number of ways to place the dividers is the same as the number of distinct subsets of cardinality $k-1$ from a set of size $k + n - 1$. □

2.5.3 Output multisets

Definition 2.5.7 (Multiset rules)**.** *Let* $|Q| = n$*. Suppose* $R_{j,k}$ *is a weak order over* $\mathcal{N}_{j,k}$*, for each* $j, k \in \{1, \ldots, n\}$*. We define a rule* $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ *as follows. Choose* $j = \max_{q \in Q} \mathsf{s}_{\trianglerighteq}(q, \Delta A Q_0)$ *(recall Definition 2.1.3). The counter* j *thus refers to the number of distinct 'levels' of* $\trianglerighteq$*. Choose* $k = |Q_0|$*. For an action* α*, define the* output multiset $X_\alpha \in \mathcal{N}_{j,k}$ *as having multiplicity function:*

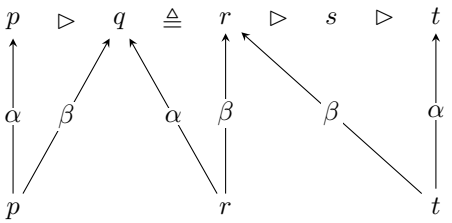
$$\mu(x) = \mid \{q \in Q_0 \mid \mathsf{s}_{\trianglerighteq}(\Delta \alpha q) = x\} \mid$$

I.e., the output multiset counts how many different outputs there are at different levels of the output states. Let $\alpha \succcurlyeq \beta$ *iff* $X_\alpha R_{j,k} X_\beta$*.*

Refer to Figure 2.12 for an example of output multisets.

Proposition 2.5.8 (Characterization of multiset rules)**.** *Any rule that satisfies Axioms 2.3.11, 2.4.3, 2.4.7, 2.4.1 and 2.4.10—that is, weak-pairwise-transition independence, belief-state symmetry, action-input symmetry, action symmetry and invariance between isomorphic possible outputs—can be expressed as a multiset rule. Conversely, such a multiset rule satisfies these axioms.*

Figure 2.12 Two output multisets.



For the setup to the left,

$$X_\alpha = \{\!\{0, 2, 3\}\!\}$$

$$X_\beta = \{\!\{2, 2, 2\}\!\}$$

Proof. This is mostly straightforward, if slightly tedious. First, we check that any multiset rule satisfies these axioms. For weak-pairwise-transition independence: the output sets of two actions are not affected by changing the transition function on *other* actions. Similarly, for belief-state symmetry and action-input symmetry the label of the input is completely irrelevant to the output set. Action symmetry is also clearly satisfied. Finally, invariance between isomorphic possible outputs precisely requires the reduction of output sets to the levels as carried out in the rule. That is, if we preserve preference relations between possible outputs, i will remain the same size and all the actions will have the same output multisets.

For the other direction, suppose we have a rule F. Suppose we are in the non-trivial case where we have at least two actions, $A = \{\alpha^*, \beta^*, \dots\}$. We need to define a ranking of multisets $R_{j,k}$ for each $j, k \in \{1, \dots, n\}$ that expresses F. Take an arbitrary pair of multisets $X, Y \in \mathcal{N}_{j,k}$. We will describe a paradigm setup that determines the ranking between these multisets. List the elements of X as $x_1, x_2, \dots, x_k$ with $x_i \geq x_{i+1}$ and of Y as $y_1, \dots, y_k$ with $y_i \geq y_{i+1}$. List the elements in Q (arbitrarily) as $q_1, \dots, q_n$. Set $Q_0^* = \{q_1, \dots, q_k\}$. Define $\trianglerighteq^*$ as [$q_i \trianglerighteq^* q_{i+1}$ for all i] and [$q_{i+1} \trianglerighteq^* q_i$ iff $i \geq j$]. Take transition function Δ^* with, minimally:

$$\begin{array}{llll} \Delta^*\alpha^* q_i = x_i & \text{and} & \Delta^*\beta^* q_i = y_i & \text{for } 1 \leq i \leq j \\ \Delta^*\alpha^* q_i = q_1 & \text{and} & \Delta^*\beta^* q_i = q_1 & \text{for } i > j \end{array}$$

Note α^* and β^* are *fixed* actions. For $F(\Delta^*, \trianglerighteq^*, Q_0^*) = \succcurlyeq^*$, simply define $X R_{j,k} Y$ if and only if $\alpha^* \succcurlyeq^* \beta^*$.

It remains to check that this multiset rule accords to the original rule F. Take an arbitrary setup $\mathsf{F}(\Delta, \trianglerighteq, Q_0,) = \succcurlyeq$ and arbitrary actions α, β such that $\alpha \succcurlyeq \beta$. Now, consider the paradigm setup according to the two multisets X_α and X_β. We want to show that $\alpha^* \succcurlyeq^* \beta^*$. Firstly, by belief-state symmetry we can permute the states from Q_0 to Q_0^* (which may change Δ):

$$\mathsf{F}(\Delta_1, \trianglerighteq, Q_0^*) = \mathsf{F}(\Delta, \trianglerighteq, Q_0)$$

Next, by action symmetry we permute the actions α and α^*, and β and β^* to get Δ_2. Thus for $\mathsf{F}(\Delta_1, \trianglerighteq, Q_0^*) = \succcurlyeq_1, \mathsf{F}(\Delta_2, \trianglerighteq, Q_0^*) = \succcurlyeq_2$:

$$\alpha^* \succcurlyeq_2 \beta^* \text{ iff } \alpha \succcurlyeq_1 \beta$$

Similarly, by invariance between isomorphic possible outputs we can modify $\trianglerighteq$ to $\trianglerighteq^*$, at the same time making sure that Δ_3 coincides with Δ^* on $Q \backslash Q_0$, such that:

$$\mathsf{F}(\Delta_3, \trianglerighteq^*, Q_0^*) = \mathsf{F}(\Delta_2, \trianglerighteq, Q_0^*)$$

By action-input symmetry we can change Δ_3 such that $\Delta_4\alpha^* q_i$ is increasing for increasing i, and the same for β^*:

$$\mathsf{F}(\Delta_4, \trianglerighteq^*, Q_0^*) = \mathsf{F}(\Delta_3, \trianglerighteq^*, Q_0^*)$$

This then implies that $\Delta_4\alpha^* q = \Delta^*\alpha^* q$ for all $q \in Q$, and the same for β^*. Thus, finally, by weak-pairwise-transition independence: for $\mathsf{F}(\Delta_4, \trianglerighteq^*, Q_0^*) = \succcurlyeq_4$,

$$\alpha^* \succcurlyeq^* \beta^* \text{ iff } \alpha^* \succcurlyeq_4 \beta^* \qquad \square$$

Note that we require multiple rankings of multisets along two dimensions. First, concerning setups with different numbers of levels in their output, we have different rankings for multisets of different maximal widths. Second, concerning setups with different sized belief sets, we have different rankings for multisets of different cardinalities.

An alternative definition could simply utilise a single ranking on all possible multisets of cardinality $\leq |Q|$. However, such rankings are less manageable. They would exacerbate the intricacies involved in ranking multisets of objects. For instance, taking $a \rhd b \rhd c$, we would have to be able to compare $\{\!\{a, b, c\}\!\}$ with $\{\!\{a, b, b, c\}\!\}$ and $\{\!\{a, c\}\!\}$, etc. But such comparisons are unnecessary; we will never need to compare multisets of different cardinalities here as Q_0 determines the size of the output set. A more manageable simplification would be to restrict attention only along the cardinality dimension.

On the other hand, note multiset rules as defined above automatically define a family of rules for arbitrary A. One method in line with the full definition is to take a single ranking over $\mathcal{N}_{n,n}$ and use this to generate rankings over the smaller cases.

Definition 2.5.9 (Generating sub-multiset rankings)**.** *For a multiset $X \in \mathcal{N}$, an i-th level insertion adds one to all elements $x \geq i$. Taking:*

$$f(x) = \begin{cases} x & \text{if } x < i \\ x + 1 & \text{otherwise} \end{cases}$$

the i-th level insertion is defined by the function $\mathsf{L} : \mathbb{N}_n \times \mathcal{N}_n \to \mathcal{N}_{n+1}$.[12]

$$\mathsf{L}(i, X) = \{\!\{f(x) \mid x \in X\}\!\}$$

Take a ranking $R_{j,k}$ over $\mathcal{N}_{j,k}$. A sub-i-level *ranking is a ranking $R_{j-1,k}$ defined in terms of some $i \leq j$ such that:*

$$X R_{j-1,k} Y \quad \textit{iff} \quad \mathsf{L}(i, X) R_{j,k} \mathsf{L}(i, Y)$$

A sub-i-element *ranking is a ranking $R_{j,k-1}$ defined in terms of some $i \leq j$ such that:*

$$X R_{j-1,k} Y \quad \textit{iff} \quad X \uplus \{\!\{i\}\!\} \, R_{j,k} \, Y \uplus \{\!\{i\}\!\}$$

A family of rankings is sub-level consistent *if all rankings with fewer levels than n are sub-i-levels for all i. Ditto* sub-element consistent*.*

For the rest of this section our focus is typically on single multiset rankings over sets $\mathcal{N}_{j,k}$.

[12] Multiset comprehension works like set comprehension.

2.5.4 Additively representable multiset rankings

One obvious method for defining multiset rankings is to assign utilities. The following definition looks in the opposite direction.

Definition 2.5.10 (Conder et al. (2007))**.** *An order $R_{j,k}$ on $\mathcal{N}_{j,k}$ (or more generally $\mathcal{N}_j$) is additively representable if there is a $\nu : \mathbb{N}_j \to \mathbb{R}_{\geq 0}$ such that*

$$XR_{j,k}Y \quad \leftrightarrow \quad \sum_{x\in\mathbb{N}_j} \mu_X(x)\nu(x) \geq \sum_{x\in\mathbb{N}_j} \mu_Y(x)\nu(x)$$

If we replace "$\leftrightarrow$" with "$\rightarrow$", the order is almost additively representable.[13] *More generally, an order R on $\mathcal{N}$ is additively representable if there is a $\nu : \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that:*

$$XRY \quad \leftrightarrow \quad \sum_{x\in\mathbb{N}} \mu_X(x)\nu(x) \geq \sum_{x\in\mathbb{N}} \mu_Y(x)\nu(x)$$

Note the definition above requires that utilities are not negative. This is a moot point if we only compare multisets of the same cardinalities. This means any ν that maps to negative numbers can be linearly shifted to an equivalent one that does not; indeed further we can take $\min_x \nu(x) = 0$.[14]

Definition 2.5.11. *The* paradigm additive representation rule *is defined in terms of some additively representable multiset ranking $R_{|Q|,|Q|}$. For each $1< j \leq |Q|$ iteratively define $R_{j-1,|Q|}$ as the sub-j-level ranking of $R_{j,|Q|}$. For each $1 < k \leq |Q|$ define $R_{j,k-1}$ as any sub-element ranking of $R_{j,k}$.*

This definition is somewhat arbitrary, as the following proposition shows.

Proposition 2.5.12. *Additively representable rules are sub-element consistent. They are not sub-level consistent.*

Proof Sketch. To see this, recall Rule 2.1.4, and compare it with:

Rule 2.5.13 (Multiset count rule)**.** *Take the ranking on each $\mathcal{N}_{j,k}$ as the additively representable ranking with $\nu(x) = x$.*

These rules produce different results, for example in Figure 2.12. □

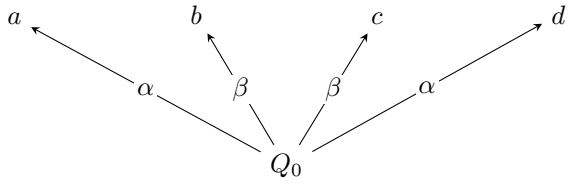
Proposition 2.5.14. *All additively representable multiset* rankings *with at least four different possible utilities are* part of *some rule that violates Axiom 2.3.15 (crossing independence).*

Proof. Take arbitrary $\nu(a) > \nu(b) > \nu(c) > \nu(d) \geq 0$. We can without loss of generality take $\nu(d) = 0$. Take two actions, α and β. We now consider two transition functions where only the outputs of Figure 2.13 are possible. That is, α can only 'get' to the top or bottom states, and β can only get to the two middle states. In particular suppose first all the states in Q_0, with $m = |Q_0|$, map by β to b. Suppose also $n < m$ states have α mapping them to a. According to Axiom 2.3.15 this is independent from a second transition

[13] Conder et al. (2007) goes on to discuss sufficient conditions for additive and almost additive representability. In general these are too strong for the analysis here.

[14] See Appendix A.2.

Figure 2.13 A diagram showing the four possible outputs from two actions for a specific type of transition function



function which changes all the states to map β to c. Thus, it suffices to show there are n and m such that $m\nu(c) < n\nu(a) < m\nu(b)$. First, choose m such that $m(\nu(b) - \nu(c)) > \nu(a)$, possible by the Archimedean principle. We can then find the necessary n. □

Note the above proof might require very large $|Q_0|$, and thus large $|Q|$. It significance is that, in general, we cannot assume that an additively representable rule will satisfy crossing independence. Other multiset rules are stronger in this respect.

2.5.5 Vector representation

Definition 2.5.15. *The* vector *representation of a multiset $X \in \mathcal{N}_n$ is:*

$$\overline{X} = (\mu(n), \mu(n-1), \ldots, \mu(1))$$

Note this definition 'puts the best elements first'. We now rank these vectors *lexicographically.*

Definition 2.5.16 (Lexicographic orderings of multisets)**.** Pessimistic*: for two vectors $\overline{x} = (x_1 x_2 \ldots x_n)$ and $\overline{y} = (y_1 y_2 \ldots y_n)$; $\overline{x} \succ \overline{y}$ iff*

for some $i \in \{1, \ldots, n\}$, for all j with $i < j \leq n$; $\quad x_j = y_j$ *and* $x_i < y_i$

Optimistic*: $\overline{x} \succ \overline{y}$ iff*

for some $i \in \{1, \ldots, n\}$, for all $j < i$; $\quad x_j = y_j$ *and* $x_i > y_i$

Proposition 2.5.17. *The lexicographic pessimistic and optimistic multiset rankings are both sub-level and sub-element consistent.*

These ranking uncomplicatedly define a whole family of rules.

Rule 2.5.18 (Lexicographic multiset rules)**.** *The* pessimistic *family of rules are those multiset rules with all rankings pessimistic multiset. The* optimistic *family of rule are those multiset rules with all rankings optimistic multiset.*

It is unsurprising that these are refinements of the set-based pessimistic and optimistic rules.

Proposition 2.5.19. *The rules of 2.5.18 satisfy crossing independence.*

Corollary 2.5.20. *The* families *of rules in 2.5.18 are not additively representable.*

We have already started discussing when multiset rules satisfy the axiom crossing independence. We move on to consider more axioms.

2.5.6 Sub-axiomatic approach

We could here attempt a full-blown axiomatic analysis of ranking *multi*sets of objects (cf. Barbera et al. (2004)). Less ambitiously, most of the following conditions are reaped from our previous crop of axioms. Precisely, these are requirements for multiset rankings which, if satisfied by each in family of multiset rankings, ensure any rule defined by this family will satisfy some previously defined axiom. For readability, in this section we also use $\succcurlyeq$ to refer to multiset rankings. For example:

Sub-axiom 2.5.21 (Multiset monotonicity)**.** *If $x > y$ then:*

$$X \uplus \{\!\{x\}\!\} \succcurlyeq X \uplus \{\!\{y\}\!\}$$

This has as its obvious strengthening the following.

Sub-axiom 2.5.22 (Multiset positive responsiveness)**.** *If $x > y$ then:*

$$X \uplus \{\!\{x\}\!\} \succ X \uplus \{\!\{y\}\!\}$$

Note we do not have, and indeed cannot have, a similar translation that solely ensures weak positive responsiveness.

Proposition 2.5.23. *Any multiset rule that satisfies weak positive responsiveness must also satisfy strong positive responsiveness.*

Proof Sketch. To see this, recall the example of Figure 2.7. This involved two setups, for both of which $a \rhd b \rhd c \rhd d$. In the left setup we get $X_\alpha = \{\!\{3, 0\}\!\}$ and $X_\beta = \{\!\{1, 1\}\!\}$. The right setup had $X'_\alpha = \{\!\{3, 0\}\!\}$ and $X'_\beta = \{\!\{2, 1\}\!\}$. The 'purpose' of *weak* positive responsiveness was to allow *both* X_β and X'_β to be indifferent to $X_\alpha = X'_\alpha$. But as we have to rank all multisets, this is impossible; we must here prefer the latter to the former: we are forced into strong positive responsiveness. □

Sub-axiom 2.5.24 (Crossing independence)**.** *For two multisets represented as vectors:*

$$(\dots, x_i, x_{i-1}, 0, x_{i-3}, \dots) \succcurlyeq (\dots, y_i, y_{i-1}, 0, y_{i-3}, \dots) \text{ iff}$$
$$(\dots, x_i, 0, x_{i-1}, x_{i-3}, \dots) \succcurlyeq (\dots, y_i, 0, y_{i-1}, y_{i-3}, \dots)$$

and also

$$(\dots, x_i, x_{i-1}, x_{i-2}, x_{i-3}, \dots) \succcurlyeq (\dots, y_i, 0, 0, y_{i-3}, \dots) \text{ iff}$$
$$(\dots, x_i, x_{i-1} + x_{i-2}, 0, x_{i-3}, \dots) \succcurlyeq (\dots, y_i, 0, 0, y_{i-3}, \dots)$$

The following axiom is different from the others of this section in that it is drawn from external literature rather than an earlier axiom of this document.

Sub-axiom 2.5.25 (Independence of equal submultisets, Conder et al. (2007))**.**

$$X_1 \uplus Y \succcurlyeq X_2 \uplus Y \quad \rightarrow \quad \forall Z,\ X_1 \uplus Z \succcurlyeq X_2 \uplus Z$$

There are various other versions of this in the literature. This formulation is chosen to allow comparisons of multisets of the same cardinality. As such, it actually suggests a new independence axiom for the full framework.

Axiom 2.5.26 (Independence of states with identical outputs.)**.** *The outcome over two actions should be independent of possible states in which these two actions have indifferent outputs. Fix some $q \in Q_0$ and actions α, β.*[15] *Suppose $\Delta = \Delta'$ on all arguments* except *for the pairs α, q and β, q; and suppose $\Delta\alpha q \triangleq \Delta\beta q$ and $\Delta'\alpha q \triangleq \Delta'\beta q$. Then $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q_0)$.*

Proposition 2.5.27. *A multiset rule satisfies independence of states with identical outputs iff it satisfies independence of equal sub-multisets.*

What about casewise dominance? Given two multisets of outputs, the question becomes whether it is *possible* for a transition function to have one action casewise dominating another. This can happen if one multiset contains more of a highest level element than another, and for all the levels below this there are at least more higher level elements in this set.

Definition 2.5.28. *The* cumulative *vector representation of a multiset is:*

$$\overline{X}^* = (\quad \mu(n) \quad , \quad \mu(n) + \mu(n-1) \quad , \quad \ldots \quad , \quad \sum_{i=n}^{1} \mu(n) \quad)$$

Sub-axiom 2.5.29 (Casewise-dominance)**.** *If every coordinate of $\overline{X}^*$ is greater than that of $\overline{Y}^*$, and at least one is strictly greater, then $X \succ Y$.*

Proposition 2.5.30. *A multiset rule satisfies casewise dominance iff it also satisfies positive responsiveness.*

Proof. Replacing an element with a higher ranked one precisely corresponds to adding one to each coordinate with index greater than the lower element up to and including the coordinate with the index of the higher element. □

The casewise dominance requirement imposes a *lattice*[16] over the cumulative vectors. For example, any ordering over $\mathcal{N}_{4,2}$ that is part of a family satisfying casewise dominance will have to respect the lattice given in Figure 2.14. There are many possibilities for refining such lattices into a weak order. Figure 2.15 shows three such weak orders—two of which refine Figure 2.14:

(a) the pessimistic lexicographic order,

(b) an order that violates crossing independence, and

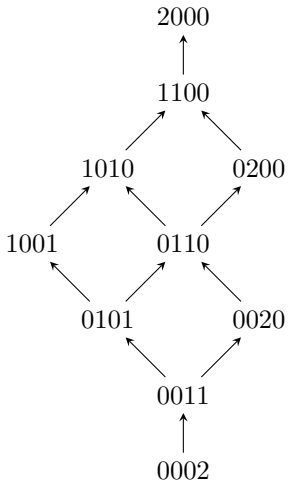
(c) an arbitrary order that is not additively representable.

We now ask: are there any other interesting rankings which still satisfy crossing independence?

[15] Note that the scope of this axiom is Q_0 and not Q. See the end of Section 3.2.

[16] I.e. every two elements have a *least join* and *most meet*, for textbook treatment see e.g. Burris and Sankappanavar (1981)

Figure 2.14 The lattice imposed by casewise dominance on the ranking of multisets of cardinality 2 with elements from a set of 4 elements. Juxtaposition is here concatenation.



The multisets are represented in (non-cumulative) vector form. Each has 2 elements, $n = 2 = |Q_0|$; and there are 4 possible elements to choose from, $k = 4 = |Q|$. The diagram gives a feeling for why casewise dominance is equivalent to positive responsiveness: each edge corresponds to replacing a single element with its covering element.

2.5.7 Regularity of risk-aversion

There are many possible extensions of a casewise dominance lattice. In many specific cases a choice can be made whether to be pessimistic, optimistic, or 'indifferentistic'. This is true even if we add crossing independence as an axiom, see Figure 2.15 (c). We now consider what happens if we require some limited consistency in these choices.

Definition 2.5.31. *Take a strict order P over a set including a and b. The element a* covers *b:*

$$a \cdot P\, b$$

if and only if aPb and there is no c such that $aPcPb$.

Sub-axiom 2.5.32 (Basic regularity)**.** *A multiset ranking is* regular *if it is consistent in its dealings with elements that cover each other. For one of $\bowtie \in \{\succ, \prec, \simeq\}$, for any elements $a, b, c \in Q$ with $a \cdot\rhd b \cdot\rhd c$, and any multiset Y:*

$$(Y \uplus \{\!\{a, c\}\!\}) \bowtie (Y \uplus \{\!\{b, b\}\!\})$$

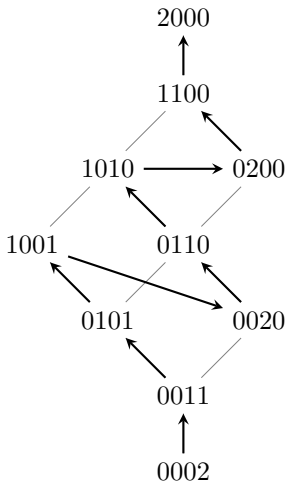
Sub-axiom 2.5.33 (Basic indifferentism)**.** *Replacing one high element and one low element with two elements that are directly in between the originals results in a set that is indifferent to the first: i.e. basic regularity with $\bowtie = \simeq$.*

This condition is incompatible with crossing independence if $|Q| > 3$. This can easily be seen by considering a fragment of the lattice of Figure 2.14, presented in Figure 2.16.

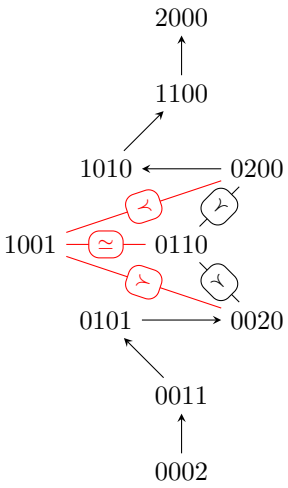
Sub-axiom 2.5.34 (Basic pessimism)**.** *Replacing one high element and one low element with two elements that are directly in between the originals results in a set that is preferred to the first. Basic regularity with $\bowtie = \prec$.*

Figure 2.15 Three refinements of casewise dominance lattices of multiset rankings. The underlying lattice in both (a) and (b) is obtained by ranking multisets of cardinality two with elements from a set of cardinality four. The lattice in (c) ranks multisets of cardinality three with elements from a set of cardinality three.

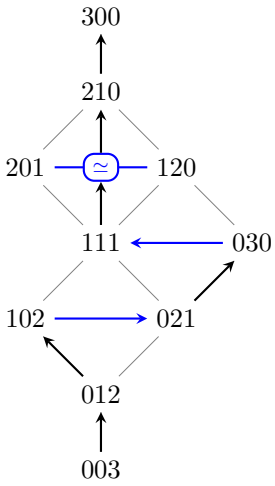
(a) The pessimistic lexicographical ordering.



(b) An ordering violating crossing independence.



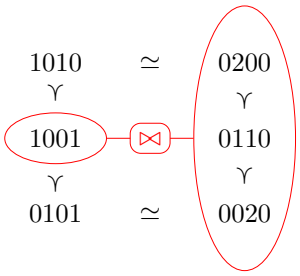
(c) A non-additively representable ordering.



The ordering of (b) can be obtained by taking utilities 5, 3, 2 and 0. The problem with crossing independence comes about at 1001: the three multisets to 1001's right should have the same relative ranking with it. The application of crossing independence is possible because of the two empty coordinates in 1001.

For (c), additive representability would require that the horizontal relations 'go in the same direction'. This would depend on whether sub-multiset (101) gets 'more score' than (020). But we can arbitrarily be indifferent, pessimistic or optimistic for each of these horizontal relations individually. Note also *any* ordering imposed over this lattice satisfies crossing independence.

Figure 2.16 Fragment of Figure 2.14.



The strict preferences on the left follow by casewise dominance. The indifferences follow by basic indifferentism. By crossing independence; *all* of vectors highlighted to the right must be either (i) strictly worse than, (ii) strictly better than, or (iii) indifferent to the middle node highlighted to the left. In case (i) the top indifference is violated, in (ii) the bottom indifference is violated, and in (iii) both are violated.

Sub-axiom 2.5.35 (Basic optimism)**.** *Replacing one high element and one low element with two elements that are directly in between the originals results in a set that the first is preferred to. Basic regularity with* $\bowtie = \succ$.

2.5.8 Characterisations

Both of the results presented in this subsection require small technical conditions.

Theorem 2.5.36. *The only multiset rules satisfying satisfying casewise dominance, independence of equal sub-multisets, simple indifference and a* technical condition $(*)$ *are expressible as the multiset count rule.*

Proof. The technical condition $(*)$ is that we require that a ranking on $\mathcal{N}_{j,2}$ for $j > 3$ is any sub-element ranking of $\mathcal{N}_{j,3}$. Without this, we cannot even ensure $1001 \simeq 0110$, as an obvious (A.2) refinement of Figure 2.14 demonstrates.[17] With $(*)$, we get the full result by Proposition 2.5.12.

It is straightforward to check that the rule satisfies the conditions.

For the proof of the other direction we write multisets as (non-cumulative) vectors $\overline{x}, \overline{y}, \overline{z}$; with coordinates for e.g. $i, j, k, (i+1), \cdots \in \mathbb{N}$. The simple count score of a vector is simply the sum of its *cumulative* coordinates. Thus, the theorem is equivalent to saying that any two vectors with the same such sum are indifferent. Take a vector of the form $\overline{0}\overline{x}\overline{0}$, where $\overline{0}$ denotes any number of 0's. We prove the claim by induction on the length of $\overline{x}$. Precisely, the inductive hypothesis (IH) is that for any two vectors with lengths of non-zero coordinates $\leq n$, if they have the same cumulative score then they are indifferent. We ignore surrounding $\overline{0}$'s when they do not figure in the following.

Base Case 1: $n = 1$: trivial.

Base Case 2: $n = 2$: also trivial; note any vector of length one *cannot* have the same cumulative sum.

Base Case 3: $n = 3$: by repeated application of simple indifference (s.ind).

Inductive step: Claim: writing $\overline{x} = xy\overline{z}wv$ with $x, v > 0$ and $\overline{z}$ possibly null:

$$xy\overline{z}wv \quad \simeq \quad (x-1)(y+1)\overline{z}(w+1)(v-1)$$

[17] As in Figure 2.14, juxtaposition is here concatenation.

The result follows from the claim by the following reasoning. Firstly note these two vectors clearly have the same cumulative sum. Thus given two arbitrary vectors with the same cumulative sum, we can iterate the procedure for each until one of $x-1$ or $v-1$ is zero, and the result follows by the induction hypothesis.

It remains to prove the claim. By the $(*)$ we only have to consider cases where $k > 2$ in $\mathcal{N}_{j,k}$.

Firstly, suppose $\overline{z}$ is null. By independence of equal sub-multisets, it suffices to show that:

$$1x01 \simeq 0(1+x)10$$

As $k > 2$, $x = y + 1$ for some $y \geq 0$. By simple indifference:

$$1(y{+}1)01 \simeq 1y20 \simeq 0(y{+}2)10 = 0(1{+}x)10 \qquad \text{as required.}$$

Now, suppose $\overline{z}$ is of length > 0. As above, by independence of equal sub-multisets it suffices to show:

$$10\overline{z}01 \simeq 01\overline{z}10$$

Write $\overline{z}$ as $x\overline{w}$. By similar considerations to above, by the independence of equal sub-multisets we can without loss of generality take $x = y + 1$:

$$10(y+1)\overline{w}01 \overset{\text{s.ind}}{\simeq} 02y\overline{w}01 \overset{\text{IH}}{\simeq} 01(y+1)\overline{w}10$$

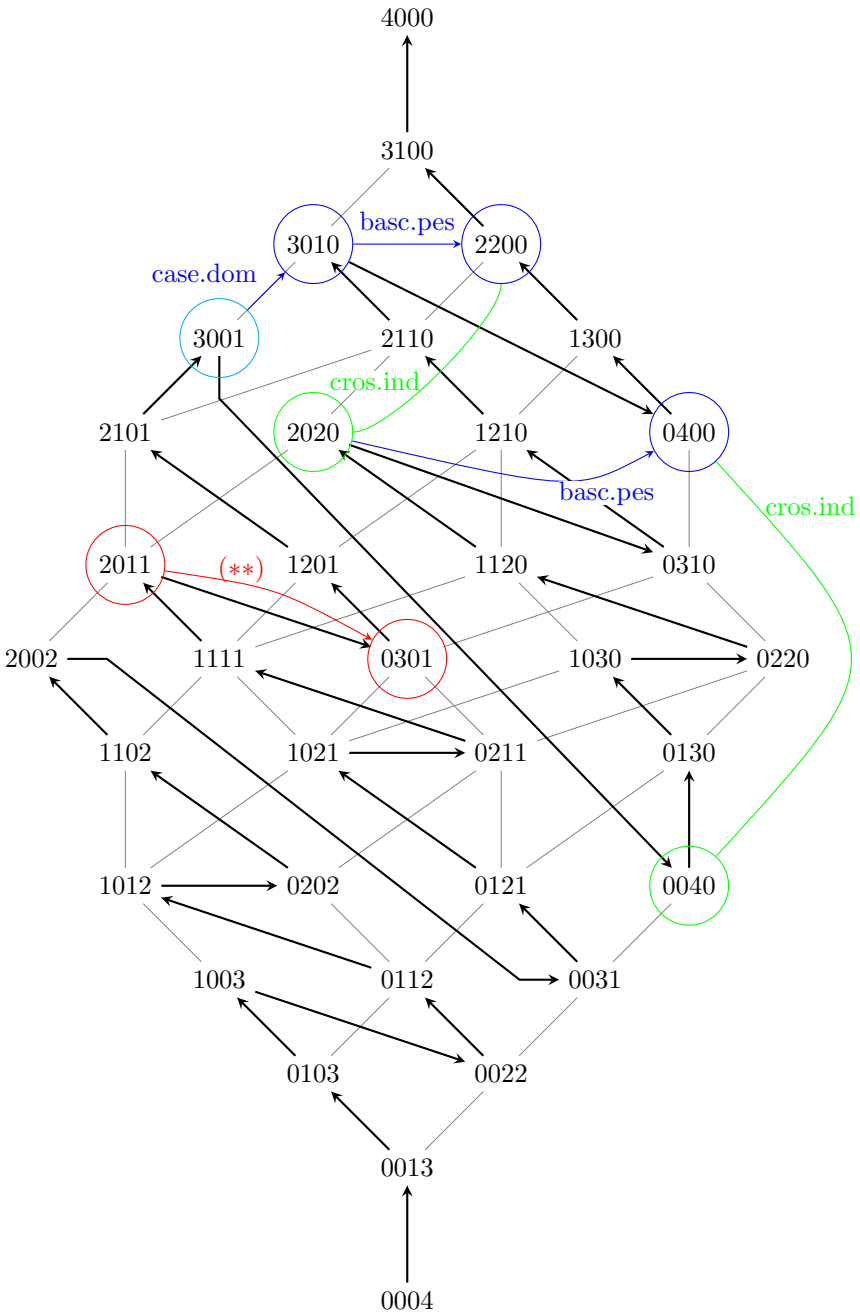
□

We have seen that crossing independence is incompatible with simple indifference. Thus, if we want basic regularity alongside crossing independence, we require either simple pessimism or simple optimism. Conversely, by adding crossing independence to either of these conditions we *almost* completely force the full pessimistic or optimistic lexicographic rule.

Theorem 2.5.37. *The only multiset rankings satisfying casewise dominance, crossing independence, basic regularity, and a technical condition* $(**)$ *are the multiset pessimistic and optimistic rules. (For* $|Q| \geq 4$*.)*

Proof. We look at the pessimistic case—the optimistic case is symmetric. We want to show $j < i$ implies $\overline{z}j\overline{y} \succ \overline{x}i\overline{y}$. This can be shown for $\overline{x}$ of length 1 or > 2, but not for length 2. The length 1 case follows immediately by casewise dominance.

Figure 2.17 Casewise dominance lattice for multisets of cardinality four over a cardinality four set; bold arrows show the pessimistic ordering; examples of the procedure of the proof of Theorem 2.5.37 are also demonstrated.



Case: length > 2. Note this implies the length of the full vector is > 3. We ignore $\overline{y}$.

$\overline{x}i$	$\preccurlyeq$	$k00\overline{0}i$	casewise dominance
	$\prec$	$k01\overline{0}(i-1)$	casewise dominance
	$\prec$	$(k-1)20\overline{0}(i-1)$	basic pessimism
$k00\overline{0}i$	$\prec$	$(k-1)11\overline{0}(i-1)$	crossing independence
	$\prec$	$(k-2)30\overline{0}(i-1)$	basic pessimism
$k00\overline{0}i$	$\prec$	$(k-2)21\overline{0}(i-1)$	crossing independence
	$\vdots$		(iteration)
$k00\overline{0}i$	$\prec$	$0(k+1)\overline{0}(i-1)$	
$k00\overline{0}i$	$\prec$	$\overline{0}(k+1)(i-1)$	crossing independence
	$\preccurlyeq$	$\overline{z}j$	casewise dominance

($**$) To get the full result, we have to *create another level*. We want to be able to ensure that $xyz\overline{w} \prec 0(x+y+1)(z-1)\overline{w}$. However, we cannot even get $xyz\overline{w} \prec (x-2)(y+3)(z-1)\overline{w}$ if $\overline{w}$ does not contain a 0. Thus, for a ranking on elements $X \in \mathcal{N}_{j,k}$, we also consider the ranking on $\mathsf{L}(i-2, X)$.[18] Then, ignoring $\overline{w}$, by crossing independence we get $xyz0 \prec 0(x+y+1)(z-1)0$ iff $0xyz \prec 00(x+y+1)(z-1)$, the latter of which is a 'length 3' case of the above. □

Hopefully the proof above intimates the strength of crossing independence in this setting. Some examples of the procedure of the proof are given in Figure 2.17. The implication of extremely strong pessimism (or optimism) was by no means obvious: for instance, it means that (100,0,1) is considered worse than (0,101,0). Without crossing independence this strong pessimism is not at all ensured by basic pessimism.

2.6 Casewise-based approach

A seemingly diametrically opposed approach to output-based methods is to consider each possible *input* on a *casewise* basis. What precisely this entails will be explored in this chapter.

2.6.1 Basic reduction to voting

One way of viewing the casewise approach is as the culmination of the "states-as-voters" metaphor. Recall Definition 2.3.1 (induced preferences), which declares that $\alpha \succcurlyeq_q \beta$ if and only if $\Delta\alpha q \succcurlyeq \Delta\beta q$.

Rule 2.6.1. *Let* $\{G_{Q_0} \mid Q_0 \in 2^Q \backslash \emptyset\}$ *be a family of aggregators with each* $G_{Q_0} : (\mathsf{wor}(A))^{Q_0} \to \mathsf{wor}(A)$. *The* induced rule *is obtained by applying the induced preferences over actions to the member of this family corresponding to the* Q_0 *of the setup.*

[18] From Definition 2.5.9. In fact, we can add the level in anywhere, then use crossing independence.

Proposition 2.6.2. *A single agent rule satisfies strong independence only if it is expressible as a induced rule. Conversely, if an aggregator is satisfies Arrovian independence, then it describes an induced-preference single agent rule that satisfies strong independence (our Axiom 2.3.12).*

However, the existence of non-independent (in the Arrovian sense) preference aggregators means that strong independence is too strong for this approach. On the other hand, crossing independence is too weak. Recall it is satisfied by the pessimistic multiset rule.

Example 2.6.3. *The pessimistic multiset rule is not inducible from any preference aggregator. This is obvious, see Figure 2.18.*

Figure 2.18 Two setups that no induced rule can distinguish.

Axiom 2.6.4 (Casewise independence)**.** *For any state, changing the output of the transition function without crossing output states* from this state *does not change the outcome. For each state $q \in Q$, for all actions α, β, suppose:*

$$\Delta\alpha q \trianglerighteq \Delta\beta q \text{ iff } \Delta'\alpha q \trianglerighteq \Delta'\beta q$$

Require:

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q_0)$$

This axiom is tailored to characterise the induced-preference rules, and hopefully it is clear that this is the case. Note that we cannot represent output-based rules in this manner.

Proposition 2.6.5. *There are multiset rules that violate casewise independence. Conversely, typically induced rules will not be expressible as multiset rules.*

Proof. For the first part refer again to Figure 2.18 as an obvious example that the pessimistic multiset rule violates casewise independence. For the second, see the right setup in Figure 2.19. □

This is in line with Endriss (2013). We make some further observations about the different situations that each approach seems to be able to deal with. In the output-based approach, there are rules that prefer either a risky action that might lead to a good output or a safe one that always leads to a mediocre output. In contrast the casewise-based approach can tell 'how often' one action will be better than another, which may not be detectable from the outputs. See Figure 2.19. Still, although the casewise-based approach seems to able to *detect*

Figure 2.19 Two setups demonstrating that different forms of risk are detectable for the output approach and the casewise approach.

this other form of risk, it is not truly able to *quantify* it. Roughly, sensible induced rules are inherently risk-averse. It seems reasonable that one might want to perform an action that is 'less likely' to be better than another, if those cases where it *is* better are *much* better. To allow for such risk-loving behaviour it seems we must go beyond the states-as-voters metaphor.

2.6.2 Casewise scoring functions

One verbal description of the casewise approach may be:

> Look at each case in isolation. Develop some measure or structure (e.g. induced preferences) over the actions for each of these cases. Transform (e.g. aggregate) a 'profile' of these structures into a single weak order over the actions.

The question we now ask is: how much information can we use for each "case" without violating the spirit of the approach? In each case, obviously the state itself is known, and so should the outputs of each action from that state.

Definition 2.6.6. *Let* $\mathsf{a} : (Q^Q)^A \to Q \to Q^A$ *be defined as, for all* Δ*,* q *and* α*:*

$$\mathsf{a}\Delta q\alpha = \Delta\alpha q$$

Then, given a setup $(\Delta, \trianglerighteq, Q_0)$*, the* casewise transition information *for each state* $p \in Q_0$ *is simply:*

$$\mathsf{a}\Delta p : A \to Q$$

If we are conservative, no information about which other states are believed should be available, let alone the outputs from these other states. What about the preferences over the states? Again, being conservative, we could restrict this preference ranking to the possible outputs from the state in question. If we do so, we effectively force the induced rules. However, if we are slightly more liberal with our information, we might want to allow the full preference ranking over states to be available. We now give a general procedure that utilises all of this information.

Definition 2.6.7 (Full casewise-based approach)**.** *A* case action scoring function *is a function:*

$$\mathsf{c} : Q^A \times \mathsf{wor}(Q) \times A \to \mathbb{R}$$

Take some family of aggregators for $1 \leq i \leq |Q|$*:*

$$\mathsf{H}_i : \mathbb{R}^i \to \mathbb{R}$$

The casewise action score of $\alpha \in A$ *is then* $\mathsf{d}(\alpha)$*, where:*[19]

$$\mathsf{d}(\alpha) = \mathsf{H}_{|Q_0|}\left(\left(\mathsf{G}\left(\mathsf{a}\Delta p, \trianglerighteq, \alpha\right)\right)_{p \in Q_0}\right)$$

Finally, take some weak order $R \in \mathsf{wor}(\mathbb{R})$*, and let* $\alpha \succcurlyeq \beta$ *iff* $\mathsf{d}(\alpha) R \mathsf{d}(\beta)$*. This defines a* full casewise rule*.*

Proposition 2.6.8. *The full casewise rules subsume both the induced rules and the output-based rules.*

Proof Sketch. For induced rules this is obvious. For output-based: assign each level in $\rhd$ a prime number. In any given case, score each action with the prime number it outputs. For the aggregation, simply multiply. By prime factorisation we know the outputs from each action, and we can rank the multisets accordingly. □

For another simple, but more concrete example, it easy to represent the pessimistic multiset rule by a *full* casewise rule. Simply assign increasing scores along with $\rhd$, for the aggregation take the minimum, and use the natural ordering on $\mathbb{R}$. However, full casewise rules go beyond both induced and output-based rules.

Example 2.6.9 (Minimizing regret)**.** *Let the case score of an action be the maximum number of levels between its output and the worst output of the case-state. For the aggregation, take the maximum of these. Use the natural ordering on* $\mathbb{R}$*. This is a risk-loving rule, in that for the right setup of Figure 2.19 it advises performing* α*.*

We note, however, that this example is implicitly relying upon cardinal information.

2.7 Chapter summary

In this chapter we have looked at a single agent model for ranking actions under uncertainty. This started by looking at desirable axioms, some of which were immediately suggested by the shape of the rule, others of which were imported from social choice. Some of these appear to be novel, as we will clarify in Chapter 3. One family of axioms we focused on were the symmetry axioms. In strengthening one of these we intimated that we might as well allow the indifferences of weak orders, rather than sticking to linear orders. This axiom and others characterised our first approach to rules: the output based approach. For this approach we further provided characterisations of concrete rules. Finally, we looked at the casewise-based approach. We considered the different circumstances a weak interpretation of this can deal with in comparison to the output-based approach, and showed that a stronger interpretation means

[19] Note for full clarity we would have to subscript $\mathsf{d}(\cdot)$, as it depends upon many factors.

that this approach actually subsumes the output-based approach. Along the way a number of interesting issues were raised.

So far as I am aware, restricting the domain *on different subspaces for different axioms* would be a novel generalisation of traditional voting theory. The uncertainty set can also be utilised to draw parallels with traditional problems in voting theory comparing profiles with different voters, such as cloning and consistency (see Appendix A.1) requirements. Other connections with traditional voting theory will be discussed in the next chapter.

We focused instead on the main two approaches to rules suggested by Endriss (2013): the output-based and casewise-based approaches. The output-based approach lead naturally to considerations on the rankings of multisets, upon which I feel there is a lot more work to be done, in comparing across families of multiset rankings, and further in simply defining rules. In particular, the necessity for the very particular technical conditions in the characterisation results needs to be investigated. Extending to the infinite case may provide some ideas, and can be done for both and separately with the width and depth of the multisets. Finally, as will be discussed more in the next chapter, I feel there is much scope for discussion on rules which take a more nuanced route in between severe pessimism and unrelenting optimism.

For the casewise-based approach, I briefly presented a new way of considering it that actually makes it a superset of the output-based approach. How satisfying this version of the approach is is perhaps a matter of taste, but the more powerful framework will allow for new descriptions of rules, some of which may interestingly satisfy some of the axioms defined previously. Finally, note that there are even more powerful approaches that take all of the information in a setup into account, that it may well be worth attempting to harness.

Chapter 3

Positioning the model

This chapter tries to find a suitable position for the model. First we consider traditional decision theory under uncertainty in Section 3.1. The model is intended to be ordinal, so in Section 3.2 we move on to specifically qualitative interpretations of this, which may be considered direct parallels to the single agent case. There are also less direct parallels with social choice theory, specifically when interpersonal comparisons are admitted, that will be explored in Section 3.3. Connections will then be drawn out with ranking both sets (Section 3.4) and multisets (Section 3.5) of objects. We look at some possibilities for the full multiagent model in Section 3.6. We conclude with Section 3.7

3.1 Traditional decision theory under uncertainty

When deciding under uncertainty, a theory could either tell you what you *should* do (prescriptive), or tell you what you *will* do (descriptive). Single agent rules may be directly interpreted as the former. General prescriptive approaches date back at least to Pascal's wager, which determined that belief in God was the correct bet. Another early problem, still disputed, is the St. Petersburg Paradox. Descriptive accounts come more recently, following an article by Wald (1939) that brings decision theory into the 'modern' era. This was continued by Savage (1972), who takes an axiomatic approach rather different to ours.

Savagean Postulates

The aims of Savage's *The Foundations of Statistics* (1972) are evidently going to be more ambitious than ours. We start with a model, then consider sensible axioms. A (loose) interpretation of Savage's approach is rather: start with axioms, then try to develop a model.[20] His intention is to found statistics on Bayesian ground—Savage et al. (1961) provides some exposition, without the full detail of his (1972).

It is still worthwhile to consider what we will term the Savagean Postulates, as they are so foundational for decision theory under uncertainty. Thus they often form the comparison basis for the more qualitative approaches, especially

[20]Though it is perhaps unclear how much he necessarily imports through definitions and the general setup of the framework. See the next Section 3.1.

those which more directly relate to our case. The translation of the Savagean Postulates into our framework is actually fairly straightforward: most of the concepts and objects are similar. Where possible, we continue with our previous notation and definitions rather than those of Savage; so we once again here have a set of actions $A = \{\alpha, \beta, \dots\}$ and a set of states $Q = \{p, q, \dots\}$. In one sense, the 'target' of the model is the same: a relation $\succcurlyeq \subset A \times A$, only, for Savage, this is taken as given at the start of the process. A more concrete difference, and one typical of decision theory in general, is that *consequences* are separate from the input states. We will make a nod to this by denoting consequences as $c, d, \dots$ Note that in our model we can simulate this division by only allowing certain states to be chosen in the possibility sets, and not allowing these sets to be outputs. On the other hand, identifying states and consequences allows for iterated runs of the model.

Savage defines *events*, which are somewhat analogous to our belief sets.

Definition 3.1.1. *An* event E *is a subset of* Q.

An event may be considered as a common property that elements of E have that none of $Q \backslash E$ have.

An important object for our analysis is the transition function, which defines what each action does. We went further than Endriss (2013) in our treatment of this an explicit part of the model. Savage does not have an equivalent object. He does, however, have notation that allows for arbitrary combinations of actions via events:

Definition 3.1.2. *Given two actions* α *and* β*, we have a new action* $\alpha_E\beta$ *such that* $(\alpha_E\beta)q = \alpha q$ *if* $q \in E$*, and* βq *otherwise.*

This is very different from our treatment of finitely many fixed actions, with outputs determined by a transition function. Without the transition function, we denote the output of an action α in a state q simply by αq. Now we are in a position to give all the Savagean postulates.

Definition 3.1.3. *The Savage postulates* **(Pn)** *and definitions* **(Dn)** *(following his numbering) in brief:*

(P1) $\succcurlyeq$ *is transitive and complete (cf. Definition 1.2.1).*

(D1) $\alpha \succcurlyeq_E \alpha'$ *iff for all* β, $\alpha_E\beta \succcurlyeq \alpha'_E\beta$.

(D2) *For an action* α *with* $\forall q \in Q$, $\alpha q = c$, *write* c.

(D3) *An event* E *is* null *iff* $\forall \alpha, \beta$, $\alpha \succcurlyeq_E \beta$.

(P2) *All* α, α': $\alpha \succcurlyeq_E \alpha'$ *or* $\alpha' \succcurlyeq_E \alpha$.

(P3) *If* $c \succcurlyeq d$ *then for non-null* E, $c_E\alpha \succcurlyeq d_E\alpha$.

(D4) $E \sqsupseteq D$ *iff for* $c \succcurlyeq d$, $c_E d \succcurlyeq c_D d$.

(P4) *All* E, D, $E \sqsupseteq D$ *or* $D \sqsupseteq E$.

(P5) *Not all* α, β, $\alpha \succcurlyeq \beta$.

(P6) *For actions* γ *and* $\alpha \succ \beta$*; there is a partition* $(E_i)_{i=1}^n$ *with* $\alpha \succ \gamma_{E_i}\beta$ *and* $\gamma_{E_i}\alpha \succ \beta$ *for each* i.

(P7) *For α, β; if all $p \in E$, $\alpha \succ_E (\beta p)$ then $\alpha \succcurlyeq_E \beta$; conversely if all $p \in E$, $(\beta p) \succ_E \alpha$ then $\beta \succcurlyeq_E \alpha$.*

In various circumstances **(P1)** could be relaxed, and we will note some pertinent attempts to do so. The second postulate **(P2)** may seem innocuous, but in combination with **(D1)** it forms what is called the *sure thing principle*, which is by no means uncontroversial. We will look at this more in the subsection below.

In considering **(P3)** we reiterate that Savage does not start with an underlying ordering over the consequences. Thus it is somewhat forced to translate this postulate. Nevertheless it may be translated as a weak triviality axiom, ensuring that the non-null sets E contain states that are not considered impossible.

We cannot accommodate **(P4)**—a condition often explained in terms of betting on an event's occurrence—into our framework currently. Section 3.2 will consider its possible integration.

(P5) is a simple non-triviality requirement. Though **(P6)** is clearly more complex, it is not easily applicable to our fixed, finite model.

The final postulate **(P7)** requires that it every consequence of an action is better than the best consequence of another action, the first action must be better than the other, and conversely that if an action's consequences are better than another action, the first is at least as good as the second. This amounts to a very weak form of casewise dominance, so weak in fact that it will not figure in the following.

After a fair amount of work, Savage obtains his full, celebrated result: there must be a unique underlying probability distribution over the events and utility function over the consequences. A more immediate consequence of **(P1-7)** is that the set of states, and by extension, the set of actions, must be infinite—contrary to our model. This is particularly due to **(P6)**, which as with **(P7)** will not be treated in the following. More importantly, we still have had no explicit consideration of what the states, finite or infinite, *actually are*.

Criticism of Savage

Our reasons for not separating states and consequences are pragmatic: we follow Endriss (2013); and there is no particular reason to separate, whilst there may be a reason for *not* doing so as it may allow for iterated runs of the procedure[21]. Still, it may be thought that identifying states and consequences makes ordinary speech examples of the model slightly clunky, as may have been noted already in Example 2.1.1. Yet much of this awkwardness is already present in the Savagean framework, as the next examples somewhat illustrate.

A *naïve* interpretation of the sure thing principle is that, if one action always results in a better outcome than another action, then that action should be preferred. However, supposing we want to maximise utility, this ordinary language translation does not hold. Towards explicating this, Table 3.1 is taken from Jeffrey's 1982 article on the sure thing principle. Here, the best outcome for the agent is that she buys and the Republican wins, giving utility 3. Everything else equal, buying gets better results than not buying: so if she doesn't buy and the Republican wins, she gets the slightly worse payoff of 2. This suggests that the dominant strategy should be to buy. However, it is more important (going

[21] Not that we consider iterated versions of the procedure here.

Table 3.1: Actions may affect probabilities (e.u.: expected utility).

Outcomes:		Republican wins		Democrat wins		e. u.
Actions:	Buy	$u = 3$	$p = 0.1$	$u = 1$	$p = 0.9$	1.2
	Don't	$u = 2$	$p = 0.8$	$u = 0$	$p = 0.2$	1.6

by pure utilities) to the agent that the Democrat does not win: this outcome gives the lower utilities of 1 (if the agent buys) and 0 (if she doesn't buy). The crux of the issue is that the act of buying greatly increases the chance that the Democrat will win from 0.2 to 0.9. This means that the seemingly dominant action of buying has lower expected utility, 1.2 versus 1.6.

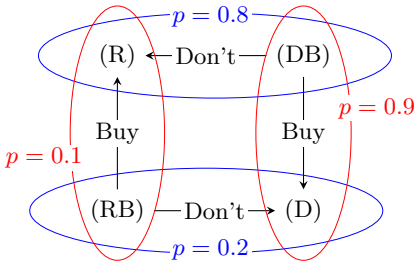
Of course, the *formal* statement of the sure thing principle *does* actually imply that dominant actions have higher expected utility. The problem above is simply not well formed with respect to the Savagean framework. Actions cannot affect probabilities directly this framework. What this means is that we need to 'translate' the natural statement of the situation above. To that end, consider the following description of four possible states, of varying awkwardness:

State 1. Republican wins (R)

State 2. Democrat wins (D)

State 3. Republican wins iff you Buy (RB)

State 4. Democrat wins iff you buy (DB)



Note there are *multiple* ways to assign probabilities in conformity with the values of Table 3.1. One is with $p(\mathrm{DB}) = 0.8$, $p(\mathrm{R}) = 0$ and $p(\mathrm{RB}) = p(\mathrm{D}) = 0.1$. Another is $p(\mathrm{DB}) = 0.7$, $p(\mathrm{R}) = 0.1$, $p(\mathrm{RB}) = 0$ and $p(\mathrm{D}) = 0.2$.

Jeffrey seems to argue that unless you have a problem with the unnaturalness of the transformation and underlying composite states, **(P2)**, and Savage's account on the whole, is sound. Of the many examples he considers[22] , particularly interesting with respect to our current work is Markowitz's problem:

> Imagine that today is your birthday; a friend presents you with a choice among three lotteries. Lottery α consists of a barrel of 2000 tickets of which 2 are marked \$1000 and the rest are blanks. Lottery β consists of another barrel of 2000 tickets of which 20 are marked \$100 and the rest are blanks. Lottery γ consists of a barrel of 2000 tickets of which 1 is marked \$1000 and 10 are marked \$100. From the chosen barrel one ticket will be drawn at random and you will win the amount printed on the ticket. Which barrel would you choose?

The claim is that, if maximising expected utility, one should rank option γ in between option α and β, as γ is an admixture of the two others. We now trans-

[22] Most of the (voluminous) criticism of the sure thing principle seems not to immediately apply to our qualitative approach. Thus we skip over Allais' problem; and note Ellsberg's problem is far too steeped in probabilities.

late the above "ebullient" statement of the problem (due originally to Alchian (1953)) into the language of multiset rankings—of Definition 2.5.7 and Definition 2.5.15. Abstracting away from the distracting monetary values, it amounts to a choice between three vectors:

Example 3.1.4 (Markowitz)**.** *Reduced to qualitative foundations: choose between the following three vectors:*

$$X_\alpha = (203) \qquad X_\beta = (031) \qquad X_\gamma(112)$$

The above claim thus becomes Sub-axiom 2.5.32. Cf. also the question of additive representability in Figure 2.15 (c). Contrary both to the claim and our basic regularity axiom, we note that, under the 'hardest'-nosed interpretation, *any* shape of preferences over α, β and γ may well be possible: one may prefer some risk to no risk, but also moderate risk to high risk; thus one may prefer γ overall. For Jeffrey, being able to work these preferences as extra utilities into the model means that you can prefer γ without violating **(P2)** and related principles. For us, it would be interesting to consider the precise different forms of possible uncertainty aversion. Some suggestions are found in Table 3.2.

Table 3.2: Possible considerations for different types of risk-aversion, possibly resulting in different rankings even though naively they might be required to be the same.

Description	Naïvely identical rankings
When risk is doubled, tripled, etc. (similar to Markowitz)	$(202) \bowtie (121)$ $(121) \bowtie (040)$ $(202) \bowtie (040)$
When there are different underlying amounts of objects	$(101) \bowtie (020)$ $(323) \bowtie (242)$ $(311) \bowtie (230)$ $(113) \bowtie (032)$
When risk occurs at different levels	$(1101) \bowtie (1020)$ $(1011) \bowtie (0201)$
When there are gaps between the high and low elements	$(11011) \bowtie (10201)$ $(11011) \bowtie (01210)$ $(10201) \bowtie (01210)$

The argument that any problem with the sure thing principle can be removed by a suitable translation—and thus that these problems really amount to problems with the naturalness of the model—is persuasive. This leaves the issue of the translation itself. Now, whilst unnaturalness of the model may well be a problem for someone taking a descriptive stance, the model itself is our object of study. Ultimately, we see no particular reason to reject **(P2)**.

Unfortunately, *none* of the rules defined in Chapter 2 currently satisfy the sure thing principle. A direct translation of **(P2)** into our framework yields a

strengthened version of Axiom 2.5.26, applied to Q instead of Q_0.[23] This is incompatible with Axiom 2.2.5. However, the next section will show that not much rearrangement is required to find a place for **(P2)**, and indeed for **(P4)**, within our model.

3.2 Qualitative decision theory

Though we eschew probability distributions, there is no particular reason not to treat **(P4)** in an *ordinal* manner.

Definition 3.2.1. *A* likelihood *ranking is a weak order* $\sqsupseteq \subseteq 2^Q \times 2^Q$.

Theories that aim for such a weakening of the full the Savagean result may be termed *qualitative decision theories*, though this term encompasses a variety of other methods. Thus also falling under this general heading are the attempted logical formalisation of Boutilier (1994); the use of sets of integers by Tan and Pearl (1994) to describe rough probabilities or utilities; and yet further attempts at founding the theory by Brafman and Tennenholtz (1996). However, these are not reconcilable with the Savagean way of thinking.

Work with explicit consideration of the Savagean postulates was initiated by Dubois and Prade (1995). Dubois et al. (1997) continue this, providing a method for "lifting" preferences to events in the following manner:

Definition 3.2.2. *The dominated states between two actions* α, β *are:*

$$[\alpha \rhd \beta] = \{q \in Q \mid \alpha q \rhd \beta q\}$$

Taking dominated states as an event, they then define the *likely dominance rule*:

Definition 3.2.3. *The likely dominance rule assigns the following ranking on actions:* $\alpha \succcurlyeq \beta$ *iff* $[\alpha \rhd \beta] \sqsupseteq [\beta \rhd \alpha]$

Let us consider some methods of deriving the ordering $\sqsupseteq$ from setups of our framework. Firstly, suppose $E \sqsupseteq D$ iff $|E \cap Q_0| \geq |D \cap Q_0|$. We note that then the likely dominance rule becomes the failed rule attempted in Example 2.3.9. Obviously, there are going to be problems here. The only suitable derived ranking that will work with likely dominance is the following: $E \sqsupseteq D$ iff $E \subseteq Q_0$.

Fargier and Perny (1999) introduce an axiom, *qualitative independence*, which is identical to our strong independence. Though this gave us the impossibility result of Proposition 2.3.14, recall that the Savagean method is partially going in the opposite direction; looking to describe $\sqsupseteq$. A preliminary result in the paper shows that strong independence, alongside **(P3)**, implies **(P2)** and **(P4)**. For one of the main conclusions, they slightly strengthened **(P5)** to **(P5')** as follows:

(P5') There are three consequences c, d, e such that $c \succ d \succ e$.

The conclusion of the paper is then:

[23] This is given in the Appendix A.1. Note **(P2)** is indeed naturally cast as an independence axiom, as evinced by work by Samet's on *The sure-thing principle and independence of irrelevant knowledge.*

Theorem 3.2.4. *Given* **(P1)**, **(P3)**, **(P5')** *and strong independence, under likely dominance we have for two actions*

$$\alpha \succ \beta \leftrightarrow \begin{cases} \exists q \in Q,\ \alpha q \rhd \beta q \\ \forall q \in Q,\ \beta q \rhd \alpha q \rightarrow (\exists p \in S,\ \{p\} \sqsupset \{q\} \wedge \alpha p \rhd \beta p) \end{cases}$$

This amounts to a partition of successive oligarchies. Here, likely dominance proceeds as follows: when comparing two actions, we first check the outputs from the first oligarchy. If one action is better in some of these outputs, and worse in none, then this action is preferred. If one action is better in some but also worse in some, the actions are indifferent. If all outputs are indifferent, we repeat the procedure for the next oligarchy, according to $\sqsupseteq$.

The comparison performed at each oligarchy above was extremely simplistic, essentially amounting to a form of unanimity.[24] However, without strong independence such a comparison is not forced. We can weaken it to one of the other forms of independence, and apply this approach to any of the rules of our current model as follows: simply take Q_0 as the first oligarchy and $Q \backslash Q_0$ as the second, and final, oligarchy. As well as integrating a simple version of **(P4)** into the model, this also means that **(P2)** can be satisfied. We can also easily extend our model to allow more oligarchies.

Indeed, we may even be able to go beyond the successive oligarchy paradigm with our weakening of strong independence. Dubois et al. (2002) choose to relax **(P1)** in order to allow more interesting likelihood relations. But it is also entirely possible that relaxing strong independence would also allow for different likelihood relations; a possibility mentioned by both Dubois et al. (2002) and Dubois et al. (2003). As the next section shows, strong independence effectively marks a return to the Arrovian framework; whereas we arguably have more information to work with.

3.3 Interpersonal comparisons of utility

Arrow's framework is explicitly ordinal, but also explicitly disallows interpersonal comparisons. In this section we show allowing such comparisons is natural for our model. Extensions of the Arrovian framework in this line have been formalised by Sen at least as far back as 1970.

We somewhat follow Roemer (1998) in the following exploration. Consider the following two methods for measuring preferences over Q:

1. as a weak order: $\trianglerighteq \subseteq Q \times Q$.

2. as a utility function: $u : Q \rightarrow \mathbb{R}$.

Let us consider the utility function as a vector in $\mathbb{R}_Q$. Clearly, a large set of utility functions correspond to a single weak order. The full set can be obtained by closing under coordinate-wise application of the set of strictly increasing functions $f : \mathbb{R} \rightarrow \mathbb{R}$.[25]

[24] Note that even this simplistic rule is only possible because the equivalent of our casewise dominance is not required.

[25] A function f is strictly increasing if $x > y$ implies $f(x) > f(y)$.

Example 3.3.1. *Let $Q = \{p, q, r\}$ and suppose $p \rhd q \rhd r$. One 'representative' utility function is $u = (2, 1, 0)$. Via strictly increasing $f(x) = x^2 + 4$, we see $u' = (8, 5, 4)$ also represents the ordering.*

Such an approach partitions the set of all utility functions into equivalence classes, one for each weak order. In a certain sense, these sets of utility functions represent an ordinal perspective.

Now, add a transition function Δ and belief set Q_0 and consider the induced preferences over actions as in Definition 2.3.1: $(\succcurlyeq_i)_{i \in Q_0}$. We represent this as a vector of utility functions $(u_i)_{i \in Q_0}$ defined for each $i \in Q_0$ as

$$u_i = (u(\Delta \alpha i))_{\alpha \in A}$$

Example 3.3.2. *Continuing Example 3.3.1, let also $A = \{\alpha, \beta\}$, $Q_0 = \{p, r\}$ and Δ as in Figure 2.1. Taking again $u = (2, 1, 0)$ we obtain $u_p = (2, 1)$ and $u_r = (0, 1)$. Applying $f(x) = x^2 + 4$ to the utilities of the states we see $u'_p = (8, 5)$ and $u'_r = (4, 5)$. Note under any such transformation of u to u'' the second coordinate of u''_p and u''_r will always be the same, and that the first coordinate cannot 'cross' this value.*

Taking the ordinal perspective on preferences over states thus naturally leads to allowing 'inter*case*' comparisons in the induced preferences over actions. Thus reinforced, the induced preference approach gains the expressivity of the output-based approach.[26] In particular, note simply transposing these utility vectors gets us the output multisets, regardless of the transformations the vectors have undergone.

There is a lot of work questioning the admittance of inter*personal* comparisons to the Arrovian framework: Bossert and Weymark (2004) provide a fairly comprehensive survey of axioms and results, whilst d'Aspremont and Gevers (2002) provide a critical review. What we have here is akin to a condition called *ordinal full comparability* by Blackorby et al. (1984) and called *invariance with respect to commonly increasing transformations* by d'Aspremont and Gevers. We note that, insofar as it practically constitutes our basic approach, we never defined an axiom that identifies this.[27]

The literature defines a large list of transformations for invariance conditions. One far stronger version is *invariance with respect to individually increasing transformations*. This corresponds to our casewise independence, and amounts to a return to the Arrovian framework.[28]

It might be also hoped that an invariance condition would line up with the intermediate crossing independence. However, this is difficult to translate, as Example 3.3.3 shows. So far as I am aware, crossing independence is a novel concept.

Example 3.3.3. *Continuing Example 3.3.2, for crossing independence, we would require $u''_p = (8, 6)$ and $u''_r = (5, 7)$ to be invariant. Further we could have any values $x > y, z > w$ in $u''_p = (x, y)$ and $u''_r(w, z)$. In terms of functions, for*

[26] It falls somewhat short of the full casewise approach sketched in Section 2.6.2, however, as it cannot detect levels without outputs in the ranking over states.

[27] We give the axiom in Appendix A.1. We also note despite the claim that this defines our approach, we did define an early rule that violates it: Rule 2.1.4.

[28] Though not an invariance condition, we also note that our weak-pairwise-transition independence corresponds to *binary independence* in d'Aspremont and Gevers (2002).

a particular group of utility vectors, we can apply any increasing function that is the identity on all coordinates except one, for all vectors.

The Arrovian framework, corresponding to casewise independence, is generally the most restrictive considered, thus has the smallest space for possible rules. Earlier we noted that we wanted crossing independence in some way to be a minimal strengthening of strong (and it turns out also casewise) independence that allowed to distinguish when different actions crossed. This is because such a minimal weakening allows for consideration of what extra rules it admits. Though extended in other directions as well, typically invariance with respect to commonly increasing transformations is seen as the next level of generality up from invariance with respect to individually increasing transformations. Thus crossing independence provides a novel intermediate position.

Admitting interpersonal comparisons is quite controversial: much effort is expended, e.g. by d'Aspremont and Gevers (2002), to look for possible justifications. Our model is the most natural application of the concepts involved that I have seen. The interpersonal interpretation is quite different, there is perhaps limited use in importing results or axioms. We are not interested in issues of egality for individuals, but in *risk*; in how risk averse or risk loving to be. Though the literature we have discussed in this section is perhaps not so relevant to this, we can draw from other sources.

3.4 Ranking sets of objects

Suggested by Endriss (2013), the original inspiration for the output-based approach above was from *ranking sets of objects*. Before focusing on the specific issue of relative uncertainty aversion, let us consider the general setting, surveyed by Barbera et al. (2004).[29] This provides a first division between sets of *mutually exclusive alternatives* and sets of *joint alternatives*. As we presume that only one consequence is possible, we clearly take the first fork. A second split then arises between *opportunity* and *uncertainty* sets: in the former, the agent is offered a final choice of the element from the set of elements; in the latter, chance picks the ultimate element. The former leads to many interesting questions about the role of freedom of choice in determining how highly ranked an opportunity set should be. There is a venerable tradition to this literature going back to work by Koopmans (1964) and Kreps (1979), continued more recently by Xu (2003). Also of note is work by Dutta and Sen (1996), which considers correspondence results between ranking opportunity sets and Arrow's theorems. However, we are here interested in uncertainty sets.

Uncertainty aversion for sets of objects

In the setting of ranking uncertainty sets, Bossert et al. (2000) attempted early axiomatisation of minmax and maxmin; and of lexicographic extensions of these; with some clarifications by Arlegi (2002). Although these rules may be classified as initially optimistic or pessimistic, they are balanced in that the next consideration is one of the opposite kind.

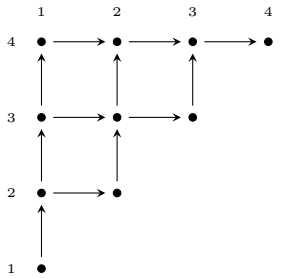
[29] Going roughly in the opposite direction to ranking sets of objects is *revealed preference theory*, which appears to be prior: see work by Sen (1971).

Even more possibilities between pure pessimism and optimism are quantified by Bossert and Slinko (2006). This starts with (a strengthened version of) a result of the pioneering work by Kannai and Peleg (1984), that amounts to the following:

> The relative ranking of two sets should only depend upon the minimum and maximum elements of those sets (3.1)

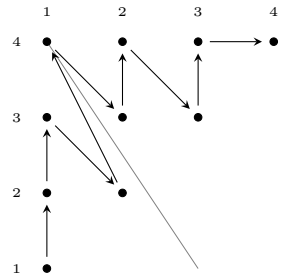
Thus they focus on ranking *pairs* of elements. With a basic monotonicity or dominance axiom, this results in an underlying lattice, as in for e.g. Figure 3.1. In order to force a refinement of this lattice they introduce a new axiom, *translation neutrality*. This states (roughly) that increasing (or decreasing) both the best and worst elements of two sets by one level should not change the relative ranking between the two sets. Though the number of possible rankings is described utilising some elegant number theory involving Farey fractions, here we will stick to the geometric interpretation. Thus, a ranking on the coordinate pairs is determined by any negative gradient, g. Consider the line drawn between two coordinates. If its gradient is positive, zero, or undefined, the relation between the pairs is determined by the dominance requirement. If two coordinates lie on the line of this slope, they are indifferent. If the line has a smaller gradient than g (greater absolutely) then the pair with the larger high element is preferred. If the line has a greater gradient than g (smaller absolutely) then the pair with the larger low element is preferred. See Figure 3.2 for an example. Note all these rankings have *uniform* risk-aversion: classifying these is seemingly a prior task to that of Table 3.2.

Figure 3.1 Lattice showing preferences on set pairs for sets with elements from $\mathbb{N}_4$.



The lattice to the left shows relations between pairs of the highest and lowest elements of sets that are forced by intuitive dominance requirements. Cf. Figure 2.14.

Figure 3.2 An almost completely pessimistic ordering of sets.



The ordering to the left is 'almost' pessimistic in that it differs from the purely pessimistic ordering by one relation: $(4,1) \succ (2,2)$. It is characterised by a slope of (say) $-2/3$.

Bossert and Slinko (2006) consciously stick to the the set-based approach. We next consider some previous work concerning ranking multisets of objects.

3.5 Ranking multisets of objects

Historically, orderings on multisets have been studied in computer science with an aim towards proving program termination. The relevant condition, *tameness*, ensures there are no infinitely decreasing chains of multisets. Such work by Martin goes back at least to 1989, where multisets are ordered as positive cones. We are, however, far more interested in the discrete cases. It appears that much of the literature has a focus on additive representability and cancellation conditions: early work concerning this was produced by Sertel and Slinko (2002) and Conder et al. (2004), and continued by Conder et al. (2007).

In a slightly more general framework, an article by Slinko (2009) seems to be amongst the most recent work. Its statement of three open problems concerning the ranking of multisets perhaps indicates that there is still a lot of work to be done before these objects are fully understood. We note that most of these papers restricted to considering orderings on $\mathcal{N}_{j,k}$ for some j and k. It appears that the added simplicity is a very desirable condition, and that attempting to rank the full space of submultisets would have been a very difficult task indeed.

I have found no direct analogues to either Theorem 2.5.36 or Theorem 2.5.37 in the literature. However, a more thorough search should also include the field of multiple criteria decision analysis, see for e.g. work by Weber (1987).

In a slightly different field, Pivato looks at extending incomplete preorders to incomplete preorders over multisets, but with the framework of each member of the multiset being the personal state of some individual.

The above sections have shown that there is still lot of work to be done in mapping out the single agent case. As such, when moving to the multiagent case it becomes hard to know in which direction to head.

3.6 The multiagent model

The issues of complexity only increase when it comes to the multiagent case. Endriss provides a suggestion to reduce this complexity: aggregate the preferences and beliefs, then apply a single agent rule. We will investigate this and similar approaches in the next chapter.

Here there are thus direct connections to the masses of work on aggregating preferences, à la Arrow. There has also been much work that would suggest many different methods for aggregating different beliefs (for example Dynamic Epistemic Logic, as by van Ditmarsch et al. (2007), or in the theories of Spohn). Also potentially applicable is the aggregation of different reference frames performed in non-monotonic reasoning, see for example Doyle and Wellman (1991).

Other ways of aggregating beliefs could certainly involve work on judgement aggregation, or rather aggregation of propositional attitudes, see for example work Dietrich and List (2008). For judgement aggregation proper, List (2012) provides a review.

As it is, the model provides an more expressive framework in which we can couch other social choice theoretic problems, such as the Ostrogorski paradox

(see Appendix B.3.1) and the paradox of multiple elections (e.g. Brams et al. (1998)). Issues of manipulation would certainly be interesting to look at here as well.

In a completely different direction, there are certainly possible connections with cooperative game theory in the vein of Shapley. The idea of cooperation is already implicit in the model, as we assume that the agents perform a single joint action. Perhaps the simplest way to connect this to cooperative games is to consider a special null action of which the interpretation is that no agreement was reached. More complicated extensions are certainly possible: we may want to allow for disjoint actions, but require that some actions require a minimal amount of participants to perform. The agents will then have to decide whether or not to form coalitions to perform these actions, even under uncertainty of what the output of the action may be. The trade off between risk, direct gains and beliefs/knowledge across different agents makes this approach particularly interesting: one agent may always like an action, whilst the other thinks it risky (either because of differences in beliefs or in preference structure), thus the first promises to compensate the second in case of failure.

3.7 Chapter summary

In this chapter I considered antecedents and connections to the model. These start with traditional decision theory, though later qualitative developments fit closer to the intended direction of our study. Indeed, the analysis of the single agent case above seems to provide a novel method in the field of qualitative decision theory. This is partly due to the connection with social choice theory, brought out particularly in the extended Arrovian framework that admits interpersonal comparisons. Allowing interstate comparisons in the qualitative framework certainly allows for more interesting oligarchy rules, and may even allow for more interesting likelihood relations that go beyond these successive oligarchies.

In the other direction, though both of the above areas of literature provide insights for our model, they do not shed much light upon the interesting question of relative risk-aversion. For our model, the simplest way to study risk aversion is to look at different ways of ranking multisets. Work on ranking sets of objects with this explicit aim has been carried out, but I have not been able to find directly equivalent work in the literature on ranking multisets of objects: our preferred expression of the possible outputs of actions. Our perspective perhaps provides a novel way of forming questions about such rankings. Note we have not even considered the possibility of different casewise forms of risk aversion here.

Thus, there are still many interesting questions to be answered, and perhaps even posed, for the single agent case. It is perhaps thus not surprising that its extension, the multiagent case, is still in an exploratory state. We now move on to this multiagent case in the next chapter.

Chapter 4

Multiagent case

4.1 Introduction

Introducing multiple agents adds yet more directions to a model which already had lots of avenues for analysis. In this chapter we consider routes of backtracking to the single agent case. One way of doing so was suggested by Endriss (2013): simply aggregate the preferences and beliefs, then apply the single agent rule. Here we investigate this, and some similar methods for simplifying the model, whilst giving an actual implementation.

Haskell

The programming language used is Haskell (Doets and van Eijck, 2004). This is a functional language: programs are executed by evaluating expressions. This may be contrasted with procedural languages such as Java or C. The deterministic—and indeed functional—nature of rules makes them good candidates for implementation with a functional language. Haskell allows us to split up the procedures into well defined expressions, which we can then naturally combine or compose. As well as implementations of the rules, we use Haskell to generate raw test data—though we note our *cultures* are not as complex as those of, say, Tideman and Plassmann (2012).

We write the code as per the literate style pioneered by Knuth (1984). This means that code is interleaved with explanatory text. As an example, the following function is part of the `Agg/Gen.hs` module.

Agg/Gen.hs

```
factorial :: (Integral n) => n -> n
factorial n = product [1..n]
```

The first line above is not strictly necessary: it explicitly gives the type of the function, which the interpreter could infer. However, it is useful to write this explicit declaration, as we can then see at a glance that `factorial` takes an integral type and returns another integral type. We will tend to be verbose in an attempt to aid comprehension. The second line gives what the function does: here, it produces a list of numbers between 1 and the argument `n`, and then multiplies them together with `product`. For the method used here to extract this interleaved code, see Appendix B.1.

Once extracted, the code is split into four files: three modules and an interface program. The first module `Agg.hs` contains basic definitions or *declarations*, the second `Agg/Rules.hs` contains rules and and other procedures that work on data, and the third `Agg/Gen.hs` concerns methods for generating test data. The interface is contained in `int.hs`. For complete export lists for each module see Appendix B.2.2.

Outline of the chapter

We first go over the full multiagent procedure and its constituent parts in the immediately following section. The main body of this chapter concerns methods for splitting up and reducing the full case to smaller procedures. Thus, Section 4.3 discuss aggregation of beliefs and preferences, Section 4.4 implements some of the single agent rules of Chapter 2, and Section 4.5 looks at specific methods of combining these to create full multiagent rules. We explain the methods used for generating test data in Section 4.6. Procedures for performing analysis on this data, including comparisons of the full multiagent rules, are given in Section 4.7. We close the chapter with a summary.

A large amount of code is relegated to Appendix B.2. This includes both instructions for extracting (B.1) and utilising (B.2.1) all the code contained in this file: `multi.tex`.

4.2 The multiagent procedure

Recall that a multiagent rule is a function that takes a setup and returns an ordering over actions (Definition 1.3.1):

$$\mathsf{F}\colon \left(Q^Q\right)^A \times \left(\mathsf{wor}(Q)\right)^J \times \left(2^Q \backslash \{\emptyset\}\right)^J \to \mathsf{wor}(A)$$

We now provide a synonym for referring to functions of such types. Note that the following is declared with a pinch of *currying*:

Agg.hs

```
type Rule = TSQ
                -> Profile (LOL' State)
                    -> Profile (BEL' State)
                        -> LOL' Action
```

The hope is that the fairly verbose style makes it clearer what each string in the above refers to. We now give explicit declarations for each of the sub-types above. First, `BEL'` represents sets of beliefs and `State` represents states, thus `BEL' State` represents beliefs over states. In fact, such sets of beliefs over states are simply implemented as lists of integers.

Agg.hs

```
type BEL' a = [a]
type State = Int
type BEL = [Int]
```

As beliefs will are typically over states, we will typically use the synonym `BEL` instead of `BEL' Int`.[30]

Occurring with two different arguments above, `LOL'` is an (unfortunate) acronym for *lists of lists*. These are used to represent the preferences over both states and `Action`s. If preferences were linear we could simply write them as *lists*. However, as per Section 2.4.2, we work with weak orders. A weak order is represented as a list of lists as follows: elements in the same sublist are indifferent, and elements in earlier sublists are preferred to those in later sublists. So in `[[2],[0,3],[1]]`: 2 is preferred to both 0 and 3, which are themselves indifferent. All three are preferred to 1.

Agg.hs

```
type LOL' a = [[a]]
type Action = Int
type LOL = [[Int]]
```

As we represent `Action`s as integers as well as `State`s, we will often use the synonym `LOL` for preferences over both. Whenever we want to make it clear that one or the other is the intended representation, we use either `LOL' State` or `LOL' Action` as required.

Though only implicit in the `Profile` type above, agents are also to be thought of as integers. A profile is a collection of homogeneous structures, one for each of a group of individuals. We utilise the library implementation of key-value mappings. Each agent corresponds to a key, and their beliefs or preferences are the value associated with that key.

Agg.hs

```
type Profile a = M.Map Int a
```

That leaves `TSQ`. This represents the transition function, here written as a *transition sequence*. It implicitly relies upon the fact that actions and states are represented as integers.

Agg.hs

```
type TSQ = [[Int]]
```

The idea is that the nth action is represented by the nth list. Within an action's sublist, the output of the action in state m is the value of the coordinate at this position. As states, individuals and actions are all composed of integers there is potential for confusion here: refer to Figure 4.1 for a diagrammatic example.

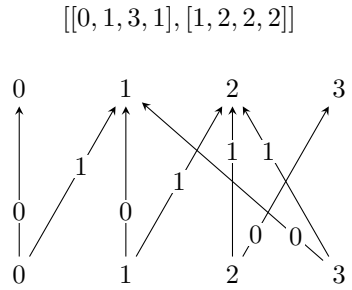
As well as causing potential confusion, the use of integers for states, actions and agents implies that we here really consider *families* of functions that range over different sized sets Q, J and A. We will at least try to keep the integers that are considered at any one time as an initial segment of the natural numbers (including zero). We now move on to consider our first family of rules, or to be precise, family of families of rules.

Early aggregation

Endriss (2013) suggested a method for reducing multiagent models to a single agent model. The basic idea is to aggregate the profiles of beliefs and preferences

[30] In general, we will add a "'" for container types, and omit the symbol when the usual container (typically `Int`) is used.

Figure 4.1 The transition function represented as a transition sequence—a list of lists of integers—and corresponding diagram.



In the diagram the starting states run along the bottom. Each starting state has two actions, 0 and 1, going out of it. These actions lead to an outcome state along the top of the diagram.

first, then apply a single agent rule. An `Aggregator` takes a profile over a structure and returns a single structure. Single agent rules are denoted `SAR`.

Agg.hs

```
type Aggregator a = Profile a -> a
type SAR = TSQ -> LOL' State -> BEL -> LOL' Action
```

Thus, a full multiagent rule is defined from two aggregators and a single agent rule by `earlyAgg`.

Agg/Rules.hs

```
earlyAgg :: Aggregator LOL -> Aggregator BEL -> SAR -> Rule
earlyAgg prefAgg belAgg sar tsq prefProf belProf =
                         sar tsq (prefAgg prefProf) (belAgg belProf)
```

An early-aggregator thus takes three arguments: an aggregator of lists of lists, an aggregator of beliefs, and a single agent rule. These then produce a full multiagent rule with the composition as described above.

Note we have still not yet described any specific rule—as mentioned above, early-aggregators are really a family of families. To remove one of these levels we now define some single agent rules and aggregators.

4.3 Aggregators

Perhaps the simplest form of aggregation, applicable to profiles over any structure, is to simply return the first structure in the profile. This amounts to making the first agent 'dictator'.

Agg/Rules.hs

```
dictatorship :: Profile a -> a
dictatorship = (snd.M.findMin)
```

For more specific rules concerning beliefs and preferences we must consider what it is that is being aggregated.

Belief aggregation

We here review a number of possibilities for belief aggregation discussed by Endriss (2013), under the heading of *uncertainty resolution*. A central notion here is that of the *support* of a state. This is simply the number of individuals who have the state in their belief set.

The support of a state may thus be thought of as the *score* of a state. Generally, we represent *scorings* as mappings. We will write `SCO' a b` when we give some keys `a` numerical scores `b`, and `SCO` for when we give a set of integers integer scores.

Agg.hs

```
type SCO' a b = M.Map a b
type SCO = M.Map Int Int
```

A profile of scorings can be aggregated by simply summing the utilities for each key, as in `sumUtility` below.[31] The function `scoWinners` will then return those keys with maximal scores. Alternatively, `scoBettAvg` returns the keys with better than average scores.[32]

Agg/Rules.hs

```
sumUtility :: (Ord a, Num b) => Aggregator (SCO' a b)
sumUtility = (M.unionsWith (+)).M.elems

scoWinners :: (Ord a, Num b, Ord b) => SCO' a b -> [a]
scoWinners sco = M.keys$M.filter (== maximum(M.elems sco)) sco

scoBettAvg :: (Ord a, Fractional b, Ord b) => SCO' a b -> [a]
scoBettAvg sco = M.keys$M.filter (> meanScore) sco
        where meanScore = (M.foldr (+) 0 sco)
                                / ((fromIntegral.M.size) sco)
```

We can now use these to help define belief aggregators. First, we work out the support for each state, which is done in the function `supportScore`. We can then apply the above functions, thus `approval` returns those states with maximal support. Approval voting has a large place in the literature on voting theory, see for e.g. work by Brams and Fishburn (2007). The alternative option, `meanBasedRule`, returns any state that receives above an average amount of support (Duddy and Piggins, 2013).

Agg/Rules.hs

```
supportScore :: (Ord a) => Profile (BEL' a) -> (SCO' a Int)
supportScore prof = sumUtility $
        M.map (M.fromList.(flip zip) [1,1..]) prof

approval :: (Ord a) => Aggregator (BEL' a)
approval = scoWinners.supportScore

meanBasedRule :: (Ord a) => Aggregator (BEL' a)
meanBasedRule = scoBettAvg.M.map fromIntegral.supportScore
```

[31] Note because we are using mappings we require that the keys are members of the `Ord` class.

[32] Note the different classes required for this: to calculate the mean we require a fractional type. Thus we also have to escape integral types in `scoBettAvg`.

Another procedure in the literature is *even-and-equal cumulative voting* (Alcalde-Unzu and Vorsatz, 2009). This equally distributes an equal weight for each agent amongst their chosen states, implemented here as `evenCumScore`. The next step is to return the states with maximal cumulative scores, which is the aggregator `evenCumWinners`. We can immediately also combine this with `scoBettAvg`, which gives us what appears to be an original, if not overly intuitive, aggregator, `evenCumAverage`.

Agg/Rules.hs

```
evenCumScore :: (Ord a, Fractional b) => Profile (BEL' a) ->
                                                    (SCO' a b)
evenCumScore prof = sumUtility $ M.map (M.fromList.distWeight)
                                       prof
            where distWeight bels = zip bels $ repeat
                                        (1/genericLength bels)

evenCumWinners :: (Ord a) => Aggregator (BEL' a)
evenCumWinners = scoWinners.evenCumScore

evenCumAverage :: (Ord a) => Aggregator (BEL' a)
evenCumAverage = scoBettAvg.evenCumScore
```

Axiomatizations of the unoriginal rules are given by Xu (2010). If the agents have, and want to retain, *knowledge*, then we can only aggregate by intersection.

Agg/Rules.hs

```
intersectionRule :: (Eq a) => Aggregator (BEL' a)
intersectionRule prof = M.foldr (intersect) someElem prof
    where someElem = snd (M.findMin prof)
```

Proposition 4.3.1. *If the* `intersectionRule` *does not return an empty set, then it coincides with both* `approval` *and* `evenCumWinners`.

Proof. Obvious. □

We list all the possibilities here in such a way that we can refer to them later by strings. Note that the dictatorship aggregator can be unproblematically included.

int.hs

```
belAggs :: [(String, Aggregator BEL)]
belAggs = [ ("approval", approval)
          , ("meanBsdR", meanBasedRule)
          , ("evnCumWn", evenCumWinners)
          , ("evnCumAv", evenCumAverage)
          , ("intersct", intersectionRule)
          , ("dictator", dictatorship)
          ]
```

In the following we will particularly focus on approval voting, as does Endriss (2013). However, we note Endriss gives reasons for preferring the:

- even-and-equal-cumulative aggregator, i.e. if we have reason to believe that those who report small sets do so because they have more accurate information; and the

- mean-based aggregator, i.e. if we do not want to exclude too many possibilities.

Preference Aggregation

Scorings can also be used for preference aggregation. Firstly, we need to translate a list of lists to a scoring. The score-value of successive levels of the list of lists is required as the first argument of `lolToSCO`.

Agg/Rules.hs

```
lolToSCO :: [Int] -> LOL -> SCO
lolToSCO vec lol = (M.fromList.concat.zipWith zip lol) repVec
    where repVec = map repeat vec
```

Borda scoring is traditionally used for linear orders.[33] A possible formulation of this is that each element's score is the number of elements it is strictly preferred to. This generalises directly to weak orders, giving us the `bordaVector` below. Cf. Definition 2.1.3. Other scoring vectors are of course possible, and perhaps even interesting. In the other direction—`scoToLOL`—there is only one possibility: in the transformation from scorings to list of lists the cardinal information is simply lost. The `borda` aggregator is then just the composition of these various translations with `sumUtility`.

Agg/Rules.hs

```
bordaVector :: LOL -> [Int]
bordaVector = tail.(scanr ((+).length) 0)

lolToBordaSCO :: LOL -> SCO
lolToBordaSCO xs = lolToSCO (bordaVector xs) xs

scoToLOL :: SCO -> LOL
scoToLOL sco | M.null sco = []
             | otherwise    =
                let (maxs,xs) = M.partition
                                  (== maximum(M.elems sco)) sco
                in  (M.keys maxs):(scoToLOL xs)

borda :: Aggregator LOL
borda = scoToLOL.sumUtility.(M.map lolToBordaSCO)
```

Borda is the typical non-Condorcet method (see Fishburn and Gehrlein (1976)). Probably the simplest Condorcet (1785) method is Copeland (Henriet, 1985). This is implemented as follows. First, for each individual create a 'matrix' with a one in coordinates where the row index is preferred to the column index, a zero when the indices are indifferent, and a minus one where the column index is preferred to the row index. `scoToMat` creates a single such matrix (as a list of lists) given a scoring. These matrices are then summed, so that the sign of each coordinate shows whether the row is preferred to the column by a majority of individuals. Apply signum to the coordinates of a particular row, then the sum of these signs gives the score of the index of the row. These steps are achieved

[33] Borda scoring is named for the 18th century French mathematician, Borda (1781). A more recent, social-choice-theoretic account of the Borda method is by Young (1974).

by `copelandSCO`, which obtains the *Copeland scores*. Composing and returning the data in a list of lists format is done by `copeland`.

Agg/Rules.hs

```
scoToMat :: SCO -> [[Int]]
scoToMat sco =
    let scoresInPos = M.elems sco          -- reduces to list
                                           -- of scores
        expand ys y = map (ordSign.(compare y))
                          ys
    in  map (expand scoresInPos) scoresInPos
            where ordSign GT = 1    -- converts an ordering to
                  ordSign LT = (-1) -- a number, cf. "signum"
                  ordSign EQ = 0

copelandSCO :: Aggregator SCO
copelandSCO profile =
    let sumMat = M.fold
                   (\sco mat -> zipWith ( zipWith (+))
                                        mat
                                        (scoToMat sco)
                                )
                   (repeat [0,0..])  -- the empty matrix
                   profile
    in  M.fromAscList $ zip [0..] $
                map (foldl' (flip ((+).signum)) 0) sumMat

copeland :: Aggregator LOL
copeland = (scoToLOL.copelandSCO.(M.map lolToBordaSCO))
```

With the general dictatorship aggregator as well, we now have three aggregators of lists of lists. We give these the following string identifiers:

int.hs

```
lolAggs :: [(String, Aggregator LOL)]
lolAggs = [ ("bord",borda)
          , ("cope",copeland)
          , ("dict",dictatorship)
          ]
```

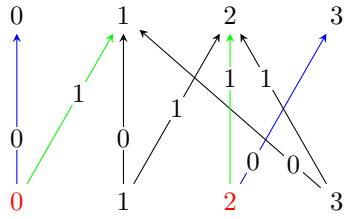
4.4 Single agent rules

The single agent rules we implement are the output-based rules of Section 2.5. As such, we need first to be able to get the outputs for each action.

Orderings: lifting to lists

The function `posOutputs` returns the output multisets, as a list of lists, for each action, given a transition sequence and a belief set. See Figure 4.2 for a demonstration. Specifically, each action is associated with an index, and the outputs from that action are the elements of the list at that index.

Figure 4.2 A diagram demonstrating the possible outputs obtained from a transition function and belief set.



For a transition sequence & belief set:

$[[0,1,3,1],[1,2,2,2]]$ & $[0,2]$

we have the following possible outcomes:

$[[0,3],[1,2]]$

Agg/Rules.hs

```
posOutputs :: TSQ -> BEL -> [[Int]]
posOutputs (act:xs) bel = map (act!!) bel : posOutputs xs bel
posOutputs []       bel = []
```

We have already seen two representations of weak orders: as list of lists (`LOL`) and scorings (`SCO`). For dealing with weak orders over lists, we introduce a third representation: as functions. In a weak order there are three preference possibilities between two elements: the first is preferred to the second (`GT`), the second is preferred to the first (`LT`), or the two are indifferent (`EQ`). A function of type `ORD` performs precisely this operation; it takes two elements as arguments, and returns `GT`, `EQ`, or `LT`. Cf. the standard library function `compare`.

Agg.hs

```
type ORD' a = a -> a -> Ordering
type ORD = Int -> Int -> Ordering
```

It will also be useful be able to translate from a list of lists to an ordering, `lolToORD`.

Agg/Rules.hs

```
lolToORD :: LOL -> ORD
lolToORD [] a b = EQ
lolToORD (xs:xss) a b| a`elem`xs && b`elem`xs = EQ
                     | a`elem`xs              = GT
                     | b`elem`xs              = LT
                     | otherwise              = lolToORD xss a b
```

We now 'lift'[34] such an ordering over elements to an ordering over lists of elements. The simplest way to do so is to apply the ordering lexicographically. The function `lexORD` utilises the fact that `Ordering` is a monoid, with `EQ` as the identity, and the binary operator `<>` returning the first argument on non-identities. Note it only compares an initial segment of a longer list, a moot point as we only compare lists of the same length per page 28.

Agg/Rules.hs

```
lexORD :: ORD -> ORD' [Int]
lexORD cmp (x:xs) (y:ys) = (x `cmp` y) <> (lexORD cmp xs ys)
lexORD cmp _      _      = EQ
```

[34]Note this is different to the "lifting" performed by Dubois et al. (1997) mentioned in Section 3.2

Strictly speaking this dictionary-style ordering is not applicable to multisets, as multisets do not come with a lexicographic order. However we can easily equip multisets with such. The two obvious versions relate to pessimism and optimism. A pessimist sorts the elements so that the worst take precedence, an optimist in the opposite manner (Cf. Definition 2.5.16).

Agg/Rules.hs

```
pesORD :: ORD -> ORD' [Int]
pesORD cmp xs ys = lexORD cmp (sortBy cmp xs) (sortBy cmp ys)

optORD :: ORD -> ORD' [Int]
optORD cmp xs ys = lexORD cmp (sortBy (flip cmp) xs)
                              (sortBy (flip cmp) ys)
```

We can also define lifted rankings that ignore the multiplicities of elements. These may then be thought of as rankings of sets of objects. The following, `domORD`, is one possibility, cf. Rule 2.5.4.

Agg/Rules.hs

```
domORD :: ORD -> ORD' [Int]
domORD cmp xs ys | minimumBy cmp xs == minimumBy cmp ys
                    = (maximumBy cmp xs)`cmp`(maximumBy cmp ys)
                 | otherwise
                    = (minimumBy cmp xs)`cmp`(minimumBy cmp ys)
```

Another general method for lifting rankings over states to rankings over multisets is to use a scoring. This approach will be embedded into the last function of this section.

Applying liftings to rules

In the opposite direction, we can 'drop' an ordering over values to apply to the keys in a mapping. First we have a helper, `indifBy`, that asks whether two elements belong to an equivalence class according to some ordering. The 'drop' itself is performed by `mapToLOL`. Note this also changes the type from an ordering into a list of list.

Agg/Rules.hs

```
indifBy :: ORD' a -> a -> a -> Bool
indifBy cmp x y = x`cmp`y == EQ

mapToLOL :: ORD' a -> M.Map Int a -> LOL
mapToLOL cmp mapping
    | M.null mapping = []
    | otherwise        =
            let isBest = indifBy cmp
                                 (maximumBy cmp (M.elems mapping))
                (xs,ys) = M.partition isBest mapping
            in  (M.keys xs):(mapToLOL cmp ys)
```

Applying this to a lifted ordering over multisets and a mapping of possible outputs gets us a single agent rule. We thus obtain either the pessimistic rule using `pesORD` or the optimistic rule using `optORD`.

Agg/Rules.hs

```
pesSAR :: TSQ -> LOL' State -> BEL -> LOL' Action
pesSAR tran lol bel = mapToLOL cmpLists $M.fromAscList$
                                zip [0..]$ posOutputs tran bel
    where cmpLists = pesORD (lolToORD lol)

optSAR :: TSQ -> LOL' State -> BEL -> LOL' Action
optSAR tran lol = mapToLOL ord . M.fromAscList . zip [0..]
                                                . posOutputs tran
    where ord = optORD (lolToORD lol)
```

The final single agent rule of this section embeds a scoring approach to ranking multisets. It amounts to Rule 2.5.13.

Agg/Rules.hs

```
indSAR :: TSQ -> LOL' State -> BEL -> LOL' Action
indSAR tran lol bel =
    let sco = lolToSCO [0,-1..] lol -- nb score vector decreasing
        listOutputs = posOutputs tran bel
        listScores =
            let outcomeScore = fromJust.(flip M.lookup) sco
            in  map (sum.map outcomeScore) listOutputs
    in  (scoToLOL.M.fromAscList) (zip [0..] listScores)
```

As has been discussed, `pesSAR`, `optSAR` and `indSAR` may be thought of as qualitative decision procedures for a single agent. We give them the following string identifiers.

int.hs

```
sars :: [(String, SAR)]
sars = [ ("pes", pesSAR)
       , ("opt", optSAR)
       , ("ind", indSAR)
       ]
```

4.5 Different routes of aggregation

We now have aggregators and single agent rules. Accordingly, we are now able to create our first reduction, by using one each of `belAggs`, `lolAggs` and `sars` as arguments in `earlyAgg`. Indeed, `earlRules` lists all the possibilities, using a helper `chain`.[35]

int.hs

```
chain :: [(String, (a->b))] -> [(String, a)] -> [(String, b)]
chain = liftM2 byCoord
            where byCoord :: (String, (a->b)) ->
                              (String, a) -> (String, b)
                  byCoord (name1, func) (name2, arg)
                              = (name1++"_"++name2, func arg)
earlRules :: [(String, Rule)]
```

[35] Strictly speaking we here only utilise the applicative property of lists, so `liftA2` would suffice. But we need to import `Control.Monad` anyway.

```
earlRules = [("earl",earlyAgg)] `chain`
                              lolAggs `chain` belAggs `chain` sars
```

Altogether, `earlRules` contains 54 elements. Restricting attention to those with `approval` as the belief aggregator brings that down to a more manageable 9. Note it is now a simple matter to define new aggregators or single agent rules and add them to the lists and thereby create new multiagent rules.

There is yet another method for creating new multiagent rules: changing the order of the reduction. We have seen how to reduce a multiagent rule to a single agent rule by aggregating the profiles first. We now look at other compositions of aggregators and single agent rules that produce multiagent rules.

Late aggregation

One option is to perform a single agent rule for each agent first, then aggregate the produced weak order over actions. The function `lateAgg` thus takes as arguments a single agent rule and an aggregator of list of lists, and composes them in this manner.

Agg/Rules.hs

```
lateAgg :: SAR -> Aggregator (LOL' Action) -> Rule
lateAgg sar agg tran pLOL pBEL = agg profActLOL
    where profActLOL = M.intersectionWith (sar tran) pLOL pBEL
```

In fact, in the most general form, we could have a different single agent rule for each individual. The implementation of this is included for completeness, we will not consider it further.

Agg/Rules.hs

```
lateAgg' :: Profile SAR -> Aggregator (LOL' Action) -> Rule
lateAgg' pSAR agg tran pLOL pBEL =
    let pSarLOL = M.intersectionWith ($tran) pSAR pLOL
        pActLOL = M.intersectionWith ($) pSarLOL pBEL
    in  agg pActLOL
```

As with early aggregation, it is now easy to combine all the previously defined aggregators and single agent rules to create a collection of *late-aggregators*.

int.hs

```
lateRules :: [(String, Rule)]
lateRules = [("late",lateAgg)] `chain` sars `chain` lolAggs
```

Non-equivalence of different routes

Proposition 4.5.1. *There are early-aggregation rules that are not late-aggregation rules, and vice versa.*

Proof. There are two directions here. We first give a late-aggregation rule that is not an early-aggregation rule. Precisely, it is the function

```
lateAgg pesSAR borda
```

identified by the string `late_pesSAR_borda` in the list `lateRules`. In particular, let us consider the following arguments:

int.hs

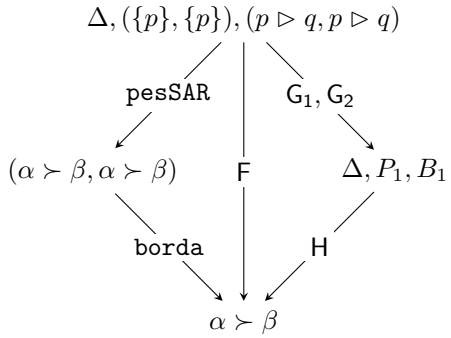
```
tsqT = [[0,1],[1,0]] :: TSQ
pLOLT = M.fromList [(0,[[0],[1]]),(1,[[0],[1]])] :: Profile LOL
pBELT = M.fromList [(0,[0]),(1,[0])] :: Profile BEL
```

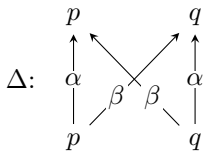
We apply these in an interactive session (see Appendix B.2.1) with

```
lateAgg pesSAR borda tsqT pLOLT pBELT
```

which returns `[[0],[1]]`: thus here action `0` is preferred to action `1`. What does this mean for any early aggregation rule? If we assume that there is an early-aggregator that gets the same output, we need three functions: a list of lists aggregator G_1, a belief aggregator G_2, and a single agent rule H; whose composition by early aggregation gets the same result. Thus, abusing different forms of notation slightly, $\mathsf{H}(\texttt{tsq2}, \mathsf{G}_1(\texttt{pLOL2}), \mathsf{G}_2(\texttt{pBEL2})) = \texttt{[[0],[1]]}$. Figure 4.3 shows the route of both late- and early aggregation applied to these arguments. Applying more arguments, and returning to previous notation, we

Figure 4.3 The early and late routes applied to a setup.





Above is the transition function Δ that corresponds to `tsq1`. Left is shown the two routes for applying late- and early-aggregation to a setup.

get the following requirements.

$$\begin{array}{lll}
\mathsf{G}_1(p \rhd q, p \rhd q) = P_1 & \mathsf{G}_2(\{p\},\{p\}) = B_1 & \mathsf{H}(\Delta, P_1, B_1) = \alpha \succ \beta^* \\
\mathsf{G}_1(p \rhd q, q \rhd p) = P_2 & \mathsf{G}_2(\{p\},\{q\}) = B_2 & \mathsf{H}(\Delta, P_2, B_1) = \alpha \simeq \beta^* \\
\mathsf{G}_1(q \rhd p, q \rhd p) = P_3 & & \mathsf{H}(\Delta, P_3, B_1) = \beta \succ \alpha^{*\dagger} \\
\mathsf{G}_1(p \triangleq q, p \triangleq q) = P_4 & & \mathsf{H}(\Delta, P_4, B_1) = \alpha \simeq \beta \ ^{\dagger} \\
 & & \mathsf{H}(\Delta, P_2, B_2) = \alpha \succ \beta \ ^{\ddagger} \\
 & & \mathsf{H}(\Delta, P_4, B_2) = \alpha \simeq \beta \ ^{\ddagger}
\end{array}$$

By the $*$ equalities P_1, P_2, and P_3 are distinct. Thus, as there are only three possible weak orders over two states, by $\dagger$ $P_4 = P_2$. But by $\ddagger$ $P_4 \neq P_2$, contradiction.

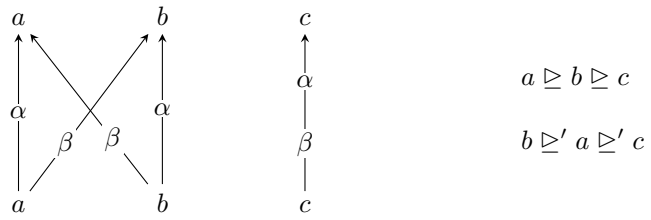
The method for giving an early-aggregation rule that is not a late-aggregation rule is similar. We consider the rule named by `earl_dict_approval_pesSAR`. The transition function Δ and two possible preference rankings, $\trianglerighteq, \trianglerighteq'$ are de-

scribed in Figure 4.4. We obtain the following list of outcomes.

$$\begin{array}{lll} F_1(\Delta, \trianglerighteq, \{c\}) = P_1 & F_2(\Delta, \trianglerighteq, \{c\}) = P_1' & G(P_1, P_1') = \alpha \simeq \beta^{*} \\ F_1(\Delta, \trianglerighteq, \{b\}) = P_2 & F_2(\Delta, \trianglerighteq, \{b\}) = P_2' & G(P_2, P_1') = \beta \succ \alpha^{*\dagger} \\ F_1(\Delta, \trianglerighteq, \{a\}) = P_3 & & G(P_3, P_1') = \alpha \succ \beta^{*\dagger} \\ F_1(\Delta, \trianglerighteq', \{c\}) = P_4 & & G(P_4, P_1') = \alpha \simeq \beta \ ^{\dagger} \\ & & G(P_1, P_2') = \beta \succ \alpha \ ^{\ddagger} \\ & & G(P_4, P_2') = \alpha \succ \beta \ ^{\ddagger} \end{array}$$

By the equations marked $*$ above we see that P_1, P_2 and P_3 are distinct. Similarly by $\dagger$, P_1, P_2 and P_3 are distinct. Finally by $\ddagger$ we see $P_1 \neq P_4$. We require four different rankings over the two actions: contradiction. □

Figure 4.4 Transition function and two possible preferences over states.



Further work would implement automated testing of whether it is possible to define a *given* aggregator in terms of early or late aggregation.

Proposition 4.5.2. *There are rules that are neither early- nor late-aggregation rules.*

Sketch proof. Here is a sketch of the required rule: first aggregate the individuals' beliefs. Then get the agents to perform a single agent rule on this group belief. Finally, aggregate the produced action preferences. It is easy to find setups which require too many distinct aggregations. □

Further routes for multiagent rules

We now implement the route suggested at the end of the previous subsection to define yet more multiagent rules out of single agent rules and aggregators. This is implemented by `earlBeliAgg'`: the name intimates the fact that here we aggregate beliefs first.

Agg/Rules.hs

```
earlBeliAgg' :: Aggregator BEL -> Profile SAR
                -> Aggregator (LOL' Action) -> Rule
earlBeliAgg' belAgg profSAR lolAgg tsq profLOL profBEL =
    let newProf = M.intersectionWith (($).($tsq)) profSAR profLOL
        newProf1 = M.map ($belAgg profBEL) newProf
    in  lolAgg newProf1
```

For simplicity, we will only consider uniform single agent rules.[36]

[36] An extension could let the agents themselves 'decide', or randomly pick a single agent rule for each.

Agg/Rules.hs

```
earlBeliAgg :: Aggregator BEL -> SAR
                -> Aggregator (LOL' Action) -> Rule
earlBeliAgg belAgg sar lolAgg tsq profLOL profBEL =
         (lolAgg . M.map (($belAgg profBEL).sar tsq)) profLOL
```

We sometimes all such routes belief-first-aggregators. For completeness we note that we could also go in the opposite direction: preference-first-aggregation.

Agg/Rules.hs

```
earlPrefAgg :: Aggregator (LOL' State) -> SAR
                -> Aggregator (LOL' Action) -> Rule
earlPrefAgg staAgg sar actAgg tsq pLOL =
        actAgg . M.map (sar tsq (staAgg pLOL))
```

We now give string identifiers for both of these reductions:

int.hs

```
beliRules :: [(String, Rule)]
beliRules = [("beli",earlBeliAgg)] `chain`
                    belAggs `chain` sars `chain` lolAggs
prefRules :: [(String, Rule)]
prefRules = [("pref",earlPrefAgg)] `chain`
                    lolAggs `chain` sars `chain` lolAggs
```

The full list of multiagent rules is then:

int.hs

```
listRules :: [(String, Rule)]
listRules = earlRules ++ lateRules ++ beliRules ++ prefRules
```

Using `length $ map fst listRules` we see that we have, thus far, 144 different multiagent rules defined. We want some way to evaluate these different rules, to see in what circumstances each may be applicable. A naïve expectation would be that the early aggregation of beliefs should 'perform well' with totally honest, mostly verdical agents. To test this, we need to be able to model agents who are at least more often than not correct about the true state of the world. We move on to generating such simple models.

4.6 Generating test data

In what follows we will mostly use uniform distributions, or, in the parlance of social choice, impartial cultures. The general method for achieving this is to first generate all the possibilities and store them in an array. The following function then generates random indices for such an array and extracts the data that these point to.

Agg/Gen.hs

```
rndsFromArr :: (Random i, A.Ix i) => A.Array i e -> StdGen -> [e]
rndsFromArr arr gen = map (arr A.!) (randomRs (A.bounds arr) gen)
```

Generating preferences

We start by showing how to generate random lists of lists. First, it would be nice to be able to count the number of possible weak orders. Recall Table 2.1; `a670` allows us to find out how many weak orders there are for much larger values. It uses the "choose" function `nCr`. This formula is not trivial: it is taken from a self published note by Kochanski (2007).

Agg/Gen.hs

```
nCr :: (Integral a, Show a) => a -> a -> a
nCr n r | n==r  = 1
        | n>r   = div (f n val) (factorial (n-val))
        where   val = max r (abs (r-n))
                f x y | x == y = 1
                      | otherwise = x * f (x-1) y

a670 :: Int -> Int
a670 0 = 1
a670 n = sum [(nCr n r) * (a670 r) | r <- [0..n-1]]
```

Because `a670` grows as quickly as it does, generating all possible weak orders for even relatively small numbers of alternatives is infeasible. Nevertheless, here is the code that does so: given input n `allLOLs` creates all weak orders over the set $\{0, \ldots, n-1\}$.

Agg/Gen.hs

```
allLOLs :: Int -> [[[Int]]]
allLOLs 0 = [[]]
allLOLs n = concatMap (allLOLs' (n-1)) (allLOLs (n-1))
    where allLOLs' :: (Eq a) => a -> [[a]] -> [[[a]]]
          allLOLs' x [] = [[[x]]]
          allLOLs' x (xs:xss) = ([x]:(xs:xss))
                              : ((x:xs):xss)
                              : map (xs:) (allLOLs' x xss)
```

Finally, we store these lists of lists in an array and apply `rndsFromArr`.

Agg/Gen.hs

```
rndLOLs :: Int -> StdGen -> [LOL]
rndLOLs nStates = rndsFromArr arr
    where arr = A.listArray (1,a670 nStates) (allLOLs nStates)
```

We here note that it would be a simple matter to prepend a filter to this list, for e.g. to restrict to single peaked preferences. See Appendix B.2.6.

Generating transition sequences

As with preferences, the number of possible transition sequences increases exponentially. Using `Control.Applicative`, the function `genActions` generates all possible actions for n states. Note there are n^n such. We then generate random such actions with `rndActs`. In order to turn this into a list of transition sequences, we could simply `unConcat` them. However, `unConcatNub` is preferable in that it ensures that no transition sequence contains two identical actions. Note this means that if the number of actions is greater than n^n then `rndTSQs` will hang.

Agg/Gen.hs

```
genActions :: Int -> [[Int]]
genActions n = iterate (liftA2 (:) [0..n-1]) [[]] !! n

rndActs :: Int -> StdGen -> [[Int]]
rndActs nSt = rndsFromArr arr
    where arr = A.listArray (1,nSt^nSt) (genActions nSt)

unConcat :: Int -> [a] -> [[a]]
unConcat n [] = []
unConcat n xs =
    let (h,t) = splitAt n xs
    in  h:(unConcat n t)

unConcatNub :: (Eq a) => Int -> [a] -> [[a]]
unConcatNub n xs = unc [] n xs n
    where unc hs 0 xs     n = hs : (unConcatNub n xs)
          unc hs i (x:xs) n | x`elem`hs = unc hs i xs n
                            | otherwise = unc (x:hs) (i-1) xs n
          unc [] i []     n = []
          unc hs i []     n = [hs]

rndTSQs :: Int -> Int -> StdGen -> [TSQ]
rndTSQs nStates nActs = unConcatNub nActs . rndActs nStates
```

Definition 4.6.1. *A transition sequence is* covering *if from every input state each output state is achievable by some action.*

The following generates minimal covering transition sequences. In these there are exactly as many states as actions.

Agg/Gen.hs

```
unConcatTranspose :: Int -> [[a]] -> [[[a]]]
unConcatTranspose n [] = []
unConcatTranspose n xs =
    let (h,t) = splitAt n xs
    in  (transpose h):(unConcatTranspose n t)

rndCovTSQs :: Int -> StdGen -> [TSQ]
rndCovTSQs n = unConcatTranspose n. rndsFromArr perms
    where perms = A.listArray (1,factorial n) (permutations [0..n-1])
```

Generating belief sets

As with lists of lists and transition sequences we could generate all the possible belief sets, `allBELs` below, then randomly choose one by its index. The helper, `allBELs'`, iteratively takes an element `x` from `[0..n-1]`, and to each constructed belief sets both prepends `xs` and returns the original set as well, utilising the applicative property of lists. This obtains the result, only also including the (unwanted) emptyset.

Agg/Gen.hs

```
allBELs' :: Int -> [BEL]
```

```
allBELs' n = foldr (\new->(<*>) [(new:),id]) [[]] [0..n-1]

allBELs :: Int -> [BEL]
allBELs n = init $ allBELs' n
```

However we here want (slightly) more complexity than a uniform distribution of belief sets. We want to model agents who can be more or less accurate, in some way.

With that aim in mind, we now treat possible worlds as complete sets of literals: equivalently, as a list of `True` and `False` values. Call these lists *Boolean descriptions*. From such Boolean description, `descToState` returns a number identifying the state.

Agg/Gen.hs

```
descToState :: [Bool] -> Int
descToState = foldl1 (+). zipWith (belT.(2^)) [0..]
    where belT x False = x
          belT _ True  = 0
```

Thus, Boolean descriptions of length k will be identified with states in the list `[0..2^k-1]`. As `0` is present in any list `[0..n]`, it makes sense to use it to represent the 'actual' world.

We model agents as 'noisy' observers of the propositions. For each proposition p, the agent can either

(i) believe p,

(ii) believe not p, or

(iii) believe p or not p.

Note an agent cannot believe neither a proposition nor its negation, as then they will not think *any* world possible. The probability of these possibilities can be assigned by `w1` and `w2` in the the *interval* $[0, 1]$, with `w1` $>$ `w2` and `w1` $+$ `w2` ≤ 1, where `w1` is the probability that an agent is correct about a proposition, and `w2` is the probability that the agent is wrong about a proposition. Given these weights and a random number `x` between 0 and 1, the function `giveBool` returns the possibilities for a given proposition. The function `rndBools` generates a random list of such possibilities.

Agg/Gen.hs

```
giveBool :: Float -> Float -> Float -> [Bool]
giveBool w1 w2 x | x < w1    = [True]
                 | x < w1+w2 = [False]
                 | otherwise = [True,False]

rndBools :: Float -> Float -> StdGen -> [[Bool]]
rndBools w1 w2 = map (giveBool w1 w2).randoms
```

From a list of possibilities for each proposition we can generate a list Boolean descriptions with `toDescs`, which then correspond to the worlds believed possible. The function `rndBELs` puts this all together to generate a list of belief sets, each of was created with the same uniform probability of getting each proposition correct.

Agg/Gen.hs

```
toDescs :: [[a]] -> [[a]]
toDescs = foldr (liftA2 (:)) [[]]

rndBELs :: Int -> Float -> Float -> StdGen -> [BEL]
rndBELs nPrps w1 w2 = map (map descToState.toDescs).unConcat nPrps
                        . rndBools w1 w2
```

Actual utilised data

Though the details are relegated to Appendix B.2.3, using the above we can clearly generate profiles for beliefs and preferences, and files containing multiple instances of each.

4.7 Analysing multiset rules

Being able to generate some data, we would now like to be able to test it upon our rules.

Decisiveness

A first test concerns how *decisive* a rule is, i.e. how often it returns a unique best action. To this end, `countDecisive` takes a list of lists and returns a triple with (i) the number of times there is a unique best action (ii) the mean size of the most preferred set and (iii) the largest size of the most preferred set.

Agg/Rules.hs

```
sizeHead :: [[a]] -> Int
sizeHead = length.head
hasUniWin :: [[a]] -> Bool
hasUniWin = (1==).sizeHead
```

int.hs

```
countDecisive :: [LOL] -> (Int, Float, Int)
countDecisive xs = (count, mean, worst)
    where count = (length.filter hasUniWin) xs
          mean  = (fromIntegral.sum.map sizeHead) xs
                        / (fromIntegral.length) xs
          worst = (maximum.map sizeHead) xs
```

I generated some test data for the `earl_bord_approval_pes` rule applied to various setups, of which some is presented in Tables 4.1 and 4.2.[37]
The results of the tables show first that generally increasing the amount of propositions increases the chance of decisiveness. This is to be expected as it increases the chance that particular actions will have different outputs, and thus be more differentiable. Second, as agents get better at correctly determining propositions, so the decisiveness decreases. This can be explained by the fact that they are then more likely to have a smaller aggregated belief set by approval,

[37] This was done using the command line with `./int data1`. See Appendices B.2.1 and B.2.4 for details.

Table 4.1: The number of decisive outcomes of `earl_bord_approval_pes` out of 10000 setups with 2 individuals and 2 actions.

Probability weights	(0.2,0.1)		(0.5,0.4)		(0.5,0.1)		(0.8,0.1)	
# propositions	1	3	1	3	1	3	1	3
# decisive	5150	9527	4930	9211	4704	8989	4682	8635

Table 4.2: The number of decisive outcomes of `earl_bord_approval_pes` out of 10000 setups with probability weights (0.2,0.1) and 2 actions.

# propositions	1	1	1	2	2	2	3	3	3
# individuals	2	5	10	2	5	10	2	5	10
# decisive	5150	5651	5899	7811	7685	7486	9527	9235	8984

and thus there is less chance for different actions to be differentiated. Third, and perhaps less obviously, that apart from the case of 1 proposition/2 states, increasing the number of individuals decreases the decisiveness.

Taking what appears to be a typical setup, we now compare decisiveness across different rules in Table 4.3. These results have perhaps too few data

rule	`earl_cope_approval_pes`	`earl_bord_approval_pes`	`earl_bord_approval_ind`	`late_pes_bord`	`late_ind_bord`	`beli_approval_ind_bord`	`pref_bord_ind_bord`	`late_ind_dict`
setup 1	7518	7450	6771	7200	7043	6410	7733	6735
setup 2	8049	8028	7492	8382	8341	7819	9076	6776
setup 3	8262	8483	8107	8894	8767	8413	9544	6711
setup 4	7341	7265	7029	7480	7459	6959	7170	6776
setup 5	8321	8291	8232	8351	8349	8268	9103	6755
setup 6	8541	8806	8797	8842	8780	8808	9634	6839
setup 7	9198	9174	7895	7937	8178	7971	8429	8079
setup 8	8986	9085	8135	8811	8814	8652	9337	8073
setup 9	8810	9124	8448	9125	9123	8973	9679	8044
setup 10	8480	8425	7876	8141	8180	8002	8159	7000
setup 11	8512	8769	8649	8807	8825	8717	9216	7006
setup 12	8582	9103	9074	9132	9084	9097	9606	6983

Table 4.3: The number of decisive outcomes for various rules out of 10000 setups of various types. Setups 1-6 have 2 propositions; setups 7-12 have 3 propositions. All have covering actions: that is, 4 for the first half and 8 for the second. Setups 1-3 and 7-9 have probability weights (0.2,0.1). Setups 1,4,7,10 have 2 individuals; setups 2,5,8 and 11 have 5 individuals; the rest have 10 individuals.

samples to be conclusive, but they strongly suggest, contra the 2 action case, that more individuals produce more decisive rules with covering transition functions. The fact that `late_ind_dict` does not show this is unsurprising as it amounts to a single agent rule performed by the first agent. More explanation would be required as to why we do not see this pattern in setups 7-9 for the early-pessimistic aggregators. Another general rule seems to be that pessimistic

aggregators are more decisive than their `ind` counterparts. There seem to be a number of other possible patterns, but larger sets of setups would be required to confirm these.

Note decisiveness does not imply that the rules are actually any good. We move on to consider that question now.

Ideal preferences

As `0` is the 'real' world, the induced preferences with this as the sole member of the belief set are, effectively, the 'real' preferences over actions. Thus `realActLOL` takes a transition sequence and a preference ranking over states, and returns a preference ranking over actions that corresponds to the ideal preference raking. Cf. Axiom 2.2.2. The same action is performed over profiles by `realActProf`.

Agg/Rules.hs

```
realActLOL :: TSQ -> LOL -> LOL
realActLOL xs lol =
    let (a0:acts) = transpose xs
    in  mapToLOL (lolToORD lol) $
                M.fromList $ zip [0..] a0

realActProf :: TSQ -> Profile LOL -> Profile LOL
realActProf xs = M.map (realActLOL xs)
```

Regardless of whether or not there is a unique best action, we can compare sets of actions using `domORD`.

Comparing rules

Thus, for each agent, we can compare the recommended actions of two different rules by that agents 'real' ranking on actions.

Agg/Rules.hs

```
cmpRulesAt :: Rule -> Rule
                -> TSQ -> Profile LOL -> Profile BEL
                -> Profile Ordering
cmpRulesAt r1 r2 tsq pLOL pBEL =
    let r1choice = last $ r1 tsq pLOL pBEL
        r2choice = last $ r2 tsq pLOL pBEL
        profORDs = M.map (domORD.lolToORD.realActLOL tsq) pLOL
    in  M.map (($r2choice).($r1choice)) profORDs
```

However, what we actually want later, is given a two lists of lists suggested by two rules, and a profile of 'true' orderings, is to return a profile of comparisons. Thus:

int.hs

```
cmpActProf :: LOL -> LOL -> Profile LOL -> Profile Ordering
cmpActProf lol1 lol2 pLOL = M.map (($head lol2).($head lol1)
                                    .domORD.lolToORD)
                                   pLOL
```

Table 4.4: Table containing comparison data for the rules `earl_bord_approval_pes`, `late_pes_bord`, `beli_approval_ind_bord` and `pref_bord_ind_bord` for four different groups of 10000 setups. All the setups have four states and covering actions. The top half have weight probabilities (0.2,0.1), the bottom (0.5,0.1). The left half have profiles with 2 individuals, the right half with 5 individuals.

		2 individuals				5 individuals			
		earl	late	beli	pref	earl	late	beli	pref
(0.2,0.1)	earl	-	1477	820	600	-	2589	707	1634
	late	1331	-	1129	861	2359	-	2213	1468
	beli	587	1014	-	888	763	2492	-	1812
	pref	1073	1493	1601	-	2014	2216	2258	-
(0.5,0.1)	earl	-	1569	436	752	-	3309	264	1881
	late	865	-	895	748	1268	-	1244	1274
	beli	271	1396	-	878	315	3265	-	1946
	pref	645	1382	952	-	1073	2689	1169	-

We use this, and the function `dataC` from Appendix B.2.4, to generate Table 4.4. Let us look at a concrete example of one of the values of the table. To create this we generated 10,000 random setups, each with transition sequences with 4 (covering) actions, a profile of list of lists over 4 states, and a profile of belief sets with 2 propositions which agents had a 0.2 probability of getting correct and a 0.1 probability of getting wrong. Each of these profiles have 2 individuals. We then generate the 'true' preferences over states for each agent. These preferences were then used to compare the suggested actions of the early- and late-aggregation rules. Altogether, 4278 agents preferred the early rule and 4196 preferred the late rule. In terms of setups, in 1477 a majority preferred the early rule, compared to 1331 where a majority preferred the late rule. In the table we only show the latter values, as these never disagree with the relative ranking of the brute number of agents.

Note that the case described above actually contained the most individual preferences between the rules of *any* of the two agent cases. As overall we have 20,000 agents for each comparison, that means that in all cases at least half of the agents were indifferent between the two rules according to our measure. It may be desirable to design a finer-grained comparison: this endeavour is initiated in Appendix B.2.5.

Because we do not seem to have many clear majorities here, we should perhaps be wary about drawing too strong conclusions from the four different cases of Table 4.4. However, some patterns can be observed, that can at least guide further investigation. First, the preference-first-aggregator does surprisingly well, performing better than the other aggregators in all case except two cases. The mechanism as to how this happens deserves investigation and explanation, particularly as our model only takes uniform cultures from the preferences. Our simple model was designed in a manner that you would expect it to favour the belief-first-aggregator, by allowing agents to correctly identify the correct world. Although the belief-first-aggregator only performs average in comparison with the other rules in general, we note some small improvements with an increase in the number of individuals. This may be because more individuals are more likely to pin down the correct world. Finally, we note that the late-aggregator

performs consistently the worst according to our model.

4.8 Chapter summary

This chapter is an implementation of the multiagent case. We started by defining data types and structures to enable this implementation in Section 4.2. Section 4.3 then described some general aggregators and Section 4.4 some single agent rules in an attempt to reduce the multiagent case to simpler procedures, four routes of which were given in Section 4.5. We looked at some methods for generating test data in Section 4.6, which was used to perform some initial analysis in Section 4.7.

This has only scratched the surface: there are still a host of questions, both following on from this implementation, and of a more theoretical nature, to be considered. We must now leave these for future work.

Conclusion

In this thesis we have investigated a model whereby multiple agents have to jointly decide upon an action to perform under uncertainty. After a brief introduction, we moved straight on to an axiomatic analysis of the single agent case. The axioms taken drew both from the shape of the model and social choice theory, the latter by treating input states as voters. We particularly considered the family of symmetry axioms, and an extension that suggested the use of weak orders is an entirely reasonable choice. This was also used is part to characterise one of two approaches we considered: the output-based approach. Within this approach we focused upon ranking multisets of objects, and gave two characterisations of specific rules, both of which required small technicalities to fully be forced through. We then moved on to second, casewise-based approach. We showed that there are (at least) two angles to this: one of which as expected is incompatible with the output-based approach, able to discern different properties of setups; the other of which, perhaps unexpectedly, subsumes the output-based approach.

The next chapter concerned our procedure's home within the general literature. Here we firmly identified the single agent case as a procedure of decision theory under uncertainty. We compared it to both the traditional model of Savage and more closely related qualitative interpretations. Some potential weaknesses of the former were seen not to apply here, as they concerned the form of the model qua model. In terms of the latter, we saw that our approach has potential to bring new insight and potentially escape some impossibility results, specifically by admitting inter'case' comparisons. These very intercase comparisons had connections drawn with interpersonal comparisons in economics literature, though we argue that they are even more plausible here than in that domain. Although many insights can be gained from the above literature, in order to help with relative uncertainty aversion and actual qualitative procedures for deciding between different forms of risk we looked at literature on ranking sets of objects. In particular we feel that ranking multisets of objects is the approach to be taking here.

Generally we saw connections to literature spanning economics, mathematics and artificial intelligence. However, concerning the multiagent case, it was less easy to find direct correlates or easily applicable related work. In the final main chapter we thus proceeded with an implementation of the full multiagent case, writing Haskell in a literate style. We now have this implementation: it remains to fully test it, and to decide upon a path down which to go with the full multiagent model.

That brings us up to here, the conclusion. Following this there are also appendices containing more work, some of which is an initiation of work still

to be done. Right now we consider some other specific directions that deserves further exploration.

We have not found evidence of any version of crossing independence elsewhere. This perhaps can be used in interpersonal comparisons as a step between ordinal full comparability and ordinal non-comparability. Alternative, it might find a place in allowing interesting likelihood relations in qualitative decision theory.

Neither have we seen generalisation of symmetry conditions that we perform before, an approach which may well deserve more attention. On a slightly different topic, I would also like to pin down why exactly it is that the technical conditions are required in our concrete characterisation results.

For another different topic, though also concerning the single agent case, there is a whole lot more work to be done concerning risk aversion or risk love. In a sense, these are difficult questions, seeing as we are in the ordinal framework. Typically one does want some kind of cardinal information to make comparisons of these kinds. Perhaps the question of how risk averse to be when considering ordinal comparisons is not connected to any reality in particular. But perhaps there is a middle ground to be etched out between the purely pessimistic or optimistic modes of thinking of pure ordinality and the utility maximising over probability distributions of cardinal information.

The multiagent case is yet more speculative at this stage. Ultimately, as ever, there are more questions than answers.

Appendix A

Single agent case: extra axioms and other miscellany

A.1 Extra axioms

There are a number of extra axioms that may be defined for the single agent case. Some of these are adaptations of traditional axioms that were not relevant to the main thrust of the work. Others differ only slightly from axioms that were already defined.

Consistency

Consistency concerns two different groups of voters whose votes get the same outcome.

Axiom A.1.1 (Disjoint uncertainty reinforcement)**.** *Two disjoint uncertainty sets that 'suggest' the same action continue to do so when combined. Take a single agent rule with*

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq \qquad \textit{and} \qquad \mathsf{F}(\Delta, \trianglerighteq, Q_0') = \succcurlyeq' \qquad \textit{and} \qquad \mathsf{F}(\Delta, \trianglerighteq, Q_0 \cup Q_0') = \succcurlyeq''$$

If $Q_0 \cap Q_0' = \emptyset$ and for two actions α, β we have both $\alpha \succcurlyeq \beta$ and $\alpha \succcurlyeq' \beta$, then we require $\alpha \succcurlyeq'' \beta$.

The necessity of taking disjoint sets of states here is demonstrated by Figure A.1.

Non-imposition conditions

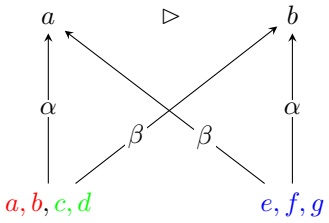
Axiom A.1.2 (Surjectivity)**.** *Every weak order over the actions is a possible output of some profile. For every $\succcurlyeq \in \mathrm{pre}(A)$, there is a profile $\Delta, \trianglerighteq, Q_0$ such that*

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$$

For a very weak version, we have:

Axiom A.1.3 (Weak non-imposition)**.** *Every strict ranking is possible for each pair of actions. For every $\alpha, \beta \in A$ there is some profile that outputs $\alpha \succ \beta$.*

Figure A.1 Example showing why simply taking the union of uncertainty sets should not preserve identical outcomes.



Taking either $\{a, b, e, f, g\}$ or $\{c, d, e, f, g\}$ as the uncertainty set, β naïvely seems to be the better option, by simply counting outputs. However, if we take the union of these (i.e. all the input states) α seems the better option.

More interesting versions would require different versions of non-imposition for specific subspaces of the full set of all possible setups.

Irrelevance of all but the output

Axiom A.1.4 (Irrelevance of all but the output)**.** *The outcome only depends upon the output sets. Over all actions α, if $\Delta\alpha Q = \Delta'\alpha Q$ then:*

$$\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q_0)$$

The sure thing principle (P2)

Cf. Axiom 2.5.26.

Axiom A.1.5 (**(P2)** equivalent)**.** *The outcome over two actions should be independent of all states in which these two actions have indifferent outputs. Fix some $q \in Q$ and actions α, β. Suppose $\Delta = \Delta'$ on all arguments* except *for the pairs α, q and β, q; and suppose $\Delta\alpha q \triangleq \Delta\beta q$ and $\Delta'\alpha q \triangleq \Delta'\beta q$. Then* $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \mathsf{F}(\Delta', \trianglerighteq, Q_0)$.

A characterising independence condition

An independence condition that arguably characterises our approach, in connection with Section 3.3.

Axiom A.1.6 (Weak crossing independence)**.** *The relative ranking of two actions only depends upon the ranking between each output and all the outputs for* both *actions. Take a single agent rule with $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ and $\mathsf{F}(\Delta', \trianglerighteq', Q_0) = \succcurlyeq'$. Require*

$$\begin{aligned}\forall p, q \in Q \,((\Delta\alpha p \trianglerighteq \Delta\beta q \quad &\leftrightarrow \quad \Delta'\alpha p \trianglerighteq' \Delta'\beta q \,) \\ \wedge (\, \Delta\alpha p \trianglerighteq \Delta\beta p \quad &\leftrightarrow \quad \Delta'\alpha p \trianglerighteq' \Delta'\beta p \,)) \\ &\rightarrow \quad (\, \alpha \succcurlyeq \beta \quad \leftrightarrow \quad \alpha \succcurlyeq' \beta \,)\end{aligned}$$

Refining preferences

Axiom A.1.7 (Refinement of state preferences)**.** *Refining state preferences cannot flip an outcome pair ranking. Take a s.a.r. with $\mathsf{F}(\Delta, \trianglerighteq, Q_0) = \succcurlyeq$ and*

$\mathsf{F}(\Delta, \trianglerighteq', Q_0) = \succcurlyeq'$, *and suppose*

$$\forall p, q \in Q \,.\, p \rhd q \to p \rhd' q$$

Then for any actions α, β

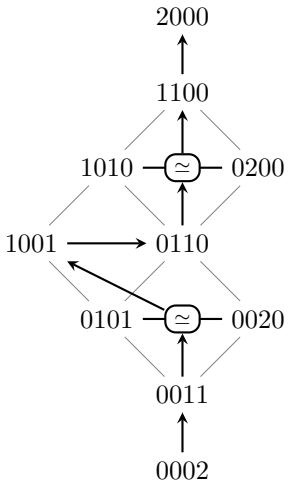
$$\alpha \succ \beta \quad \to \quad \alpha \succcurlyeq' \beta$$

This axiom is not obviously desirable. And it is even less clear how more radical changes, those involving flipping states in a preference, should affect the outcome of a rule. This is because the outcome should be heavily dependent upon the interaction of state preferences with the transition function.

A.2 Miscellany

Small results and diagrams that did not fit into the main flow of the text.

Figure A.2 Figure concerning the proof of Theorem 2.5.36: an ordering that satisfies casewise dominance, independence of equal sub-multisets and simple indifference but that is *not* expressible as the multiset count rule.



No need for negative utilities

This concerns discussion following Definition 2.5.10. Taking two multisets both of size n with total utilities:

$$u_1 + u_2 + \cdots + u_n > u'_1 + u'_2 + \cdots + u'_n$$

obviously adding a value $c = -\min_x \nu(x)$ to each score will not change the difference between these:

$$\begin{aligned}(u_1 + c) + (u_2 + c) + \cdots + (u_n + c) &= nc + u_1 + u_2 + \cdots + u_n \\ &> (u'_1 + c) + (u'_2 + c) + \cdots + (u'_n + c)\end{aligned}$$

Also this is clearly not necessarily the case if we have different cardinality multisets.

Appendix B

Multiagent case: vim script and other extra code

B.1 Vim script

Haskell, it is argued, is a good choice for implementing aggregation procedures. No argumentation is necessary to defend the use of latex for typesetting. Beneath both of these, this is being typed in (g)Vim. As an editor Vim has a lot of functionality. Specifically of interest here, it allows scripts to be written that can perform the extraction of code into separate files, in the spirit of Knuth (1984). To that end I have set up `\t` as a command to extract the code and write it into correct folders.

The scripts that give this functionality can be sourced from this very `.tex` file. They should work (at least) on Linux systems. Thus, assuming you have the file `multi.tex` open in Vim, enter

```
:let a=tempname()|exe '/scriptStart/+6;/scriptEnd/w' a|exe 'so' a
```

to source the scripts. Note this will save a temporary file. To ease sourcing of the extraction scripts, documentation is only included in the explicit code below (if you are experiencing trailing character errors on a Linux system, try `:set fileformat=unix`).

```
"-----------------------------------------
"-----------------------------------------
" Code extraction scripts for vim
"-----------------------------------------
"-----------------------------------------

"-----------------------------------------
" The keymap for actually extracting the code
"-----------------------------------------
nmap <silent> \t :echo 'working'<CR>
    \            :silent call ExtractCode()<CR>:echo 'done'<CR>

"-----------------------------------------
" Concatenates members of a list of lists
"-----------------------------------------
```

```
function! Concat(lols)
  let newlist = []
  for slist in a:lols
    let newlist += slist
  endfor
  return newlist
endfunction

"----------------------------------------
" Helper: orders and arranges the code that
" is to be written into a single file
"----------------------------------------
function! TransLPs(lol)
  call sort(a:lol)
  call map(a:lol, 'v:val[1]')
  return Concat(a:lol)
endfunction

"----------------------------------------
" The actual extraction procedure
"----------------------------------------
function! ExtractCode()
  let a:oldReg = [getreg(), getregtype('"')]  | " gets initial
  let a:origCursor = getpos(".")              | " position, to
  let a:origdir = getcwd()                    | " be returned
                                              | " to later
  cd %:p:h                        | " makes sure we write to
                                  | " the correct directory
  let a:fileAndData = {}   | " we use a dictionary
                           | " for filepath+contents

" searching through the file looking for code blocks
  call cursor(1,1)
  while search('\\be'.'gin{code}','W'  )   &&
    \   !search('NoCode'.'ExtractedAfter','nWb')
    | " the search continues until the string
    | " No Code Extracted After (without spaces)
    | " or the end of the file

" looking for and getting the data within a block
" each block has a (perhaps already seen) filepath, an
" index for insertion order, and content
    call search('}','W')           | " the file path should be
    call search('}','W')           | " before the second '}',
    execute 'normal yT{'           | " and is yanked here
    let a:filePath = getreg()
    execute 'normal f}wyt}ww'      | " yanks the index for
    let a:inputInd = getreg()      | " the input and stores
    execute 'normal m`'            | " content start marked...
    call search('\\end{code}','W') | " moves to end of content
    execute "normal y``"           | " ...and now is yanked
    let a:conts = split( getreg() ,"\n")
                | " above splits the content lines into a list
" we append the new contents in the right place in the dictionary
```

```
    let a:fileAndData[a:filePath] =
      \   extend( get(a:fileAndData,a:filePath,[])
      \         , [[a:inputInd,a:conts]]
      \         )
  endwhile

" all the data is collected in the dictionary
" it remains to write it down
  for [a:fname,a:fconts] in items(a:fileAndData)
    call WriteWithParentsFile(TransLPs(a:fconts), a:fname)
  endfor

" and finally clean up, returning as was...
  call call(function('setreg'),['""'] + a:oldReg)  |" register
  call setpos('.',a:origCursor)                    |" cursor pos
  exe "cd" a:origdir                               |" work dir.
endfunction

"----------------------------------------
" writes files into non-existing directories
"----------------------------------------
" this function, as writefile, takes a list of strings
" and a string as arguments
function! WriteWithParentsFile(fconts,fpath)
" get the path without the fname
  let a:dirname = join(split(a:fpath, '/')[0:-2], '/')
  try
    call mkdir(a:dirname, 'p')
    catch
  endtry
  call writefile(a:fconts,a:fpath)
endfunction

"---------------------------------------- scriptEnd
```

B.2 Extra Haskell code

B.2.1 Utilising the code

Once code is extracted, it can be compiled by entering `ghc --make int` into the terminal. This is necessary to use the command line interface as below, but not to access the functions interactively.

Using the command line interface

For the input/output procedures we create a simple command line interface, for which we do require compiled code. This works by entering `./int [arguments]` in a terminal, which performs the string description of the first argument identified in `dispatch`, with any subsequent arguments.

int.hs

```
main = do
```

```
    args <- getArgs
    if null args
        then do
            putStrLn "Need␣arguments"
        else do
            let (Just action) = lookup (head args) dispatch
            action (tail args)
dispatch = [ ("returnTSQs",void.returnTSQs)
           , ("returnLOLs",void.returnLOLs)
           , ("returnBELs",void.returnBELs)
           , ("returnTRUs",void.returnTRUs)
           , ("returnRULs",void.returnRULs)
           , ("returnCMPs",void.returnCMPs)
           , ("data1",data1)
           , ("data2",data2)
           , ("data3",data3)
           , ("data4",data4)
           , ("data5",data5)
           , ("data6",data6)
           , ("data7",data7)
           , ("dataD",dataD)
           , ("dataD1",dataD1)
           , ("dataD2",dataD2)
           , ("dataD3",dataD3)
           , ("dataC",dataC)
           ]
```

The strings starting with `return` require extra arguments as documented in Appendix B.2.3 below. The `datax` arguments can be used on their own, e.g. `./int data1` will generate a list of data on decisiveness. These are documented in Appendix B.2.4.

Accessing functions interactively

All of the functions defined above and below can be accessed interactively in the following manner. Run GHCI by inputting `ghci`. You can now load the top level functions for any of the files by typing, respectively: `:l *Agg`, `:l *Agg.Rules`, `:l *Agg.Gen` or `:l *int`.

B.2.2 Module declarations

Haskell modules must start with a list of functions and other definitions that are exported. Here are those for the base `Agg.hs` module.

Agg.hs

```
module Agg
( Profile (..)
, BEL   , BEL' (..)
, LOL   , LOL' (..)
, SCO   , SCO' (..)
, ORD   , ORD' (..)
, REL
, TSQ
, Aggregator (..)
```

```
, Rule  , SAR
, State , Action
)
where
```

We then give some library modules that it itself imports.

Agg.hs

```
import Data.List
import qualified Data.Set as S
import qualified Data.Map as M
```

The second module describes functions that actually do work on data.

Agg/Rules.hs

```
module Agg.Rules
( dictatorship
, approval, meanBasedRule,  evenCumWinners, evenCumAverage
, intersectionRule
, borda,    copeland
, lexORD, pesORD, optORD, domORD
, pesSAR,   optSAR, indSAR
, earlyAgg, lateAgg, earlBeliAgg,   earlPrefAgg
, realActProf
, cmpRulesAt
, hasUniWin, sizeHead
, lolToORD, lolToBordaSCO, scoToLOL
, posOutputs
)
where
import Agg
import Data.List
import Data.Maybe
import Data.Monoid
import qualified Data.Set as S
import qualified Data.Map as M
```

The final module generates data.

Agg/Gen.hs

```
module Agg.Gen
( rndLOLs, rndTSQs, rndCovTSQs, rndBELs
, a670
, genActions
, unConcat
)
where
import Agg
import System.Random
import Data.List
import Control.Applicative
import qualified Data.Map as M
import qualified Data.Array as A
```

Finally, we have the interface program. This will allow generating and writing data to disk, and then manipulating and evaluating that data, at a terminal.

int.hs

```
import Agg
import Agg.Rules
import Agg.Gen
import System.IO
import System.Environment
import System.Directory
import System.Random
import Control.Monad
import Data.List
import qualified Data.Map as M
import Data.Maybe
```

As has already been mentioned, this is not a module.

B.2.3 Folder structure for generated data

The idea here is to generate data dynamically as it is requested, with the ability to read already created files. This is achieved within a certain folder structure. All generated files are contained in a folder `data`. A subfolder indicates the number of propositions. At this level we place transition sequences, as they only depend upon the number of propositions and the number of actions, and we will not separate further into subfolders by actions. Thus transition sequences will be located in `data/[nProps]/[nActions].tsq`. If `[nActions]` is zero, the transition sequence is intended to be covering, see Definition 4.6.1. The next subfolder level indicates the number of individuals. Within this we keep profiles of both list of lists and belief sets. The former are designed to possibly have filters applied, so are written to:

`data/[nProps]/[nInds]/[filter].lol`

The latter are determined by a probability pair as per Section 4.6:

`data/[nProps]/[nInds]/([wt1],[wt2]).bel`

We also keep other files at this level, such as the outcomes of rules, as:

`data/[nProps]/[nInds]/[tsqFname].[lolFname].[belFname].[rName].rul`

Each of these files will possibly—indeed likely—contain multiple instances of the data, each written on separate lines.

As we must deal with a lot of arguments, we fix the following order: number to generate/return, number of states/propositions, number of actions, number of individuals, probability weights, filter, random generator. All functions have their arguments in this order, including those in Section 4.6 above.

Now we give the first example of generating/recovering data. Perhaps the easiest to generate, certainly the shallowest in the folder structure, are the transition sequences.

int.hs

```
ums :: (Show a) => [a] -> String    -- this is useful
ums = unlines.map show
```

```
genTSQs :: Int -> Int -> Int -> StdGen -> [TSQ]
genTSQs nToGen nPrps 0     = take nToGen . rndCovTSQs (2^nPrps)
genTSQs nToGen nPrps nActs = take nToGen
                           . rndTSQs (2^nPrps) nActs

returnTSQs :: [String] -> IO [TSQ]
returnTSQs [nToRet, nPrps, nActs] = do
    let path = "data/"++nPrps++"/"++nActs++".tsq"
    fileHere <- doesFileExist path
    if fileHere
        then do
            conts <- fmap (map read.lines) (readFile path)
            if length conts >= read nToRet
                then return (take (read nToRet) conts)
                else do
                    gen <- newStdGen
                    let nToGen = read nToRet - length conts
                    let newConts = genTSQs nToGen
                                           (read nPrps)
                                           (read nActs)
                                           gen
                    appendFile path (ums newConts)
                    return (conts ++ newConts)
        else do
            createDirectoryIfMissing True ("data/"++nPrps)
            gen <- newStdGen
            let conts = genTSQs (read nToRet) (read nPrps)
                                (read nActs) gen
            writeFile path (ums conts)
            return conts
```

The next thing to generate is the preferences. For extendability, we include a list of string identifiers of filters, now only containing the trivial filter. See Appendix B.2.6 below.

int.hs

```
lolFilters :: [(String, LOL -> Bool)]
lolFilters = [("none", \a->True)
             ]

genLOLs :: Int -> Int -> Int -> String -> StdGen -> [Profile LOL]
genLOLs nToGen nPrps nInds filtS =
    take nToGen.map (M.fromAscList.zip [0..])
    .unConcat nInds.filter cond.rndLOLs (2^nPrps)
        where cond = (fromJust.lookup filtS) lolFilters

returnLOLs :: [String] -> IO [Profile (LOL' State)]
returnLOLs [nToRet, nPrps, nInds, filt] = do
    let path = "data/"++nPrps++"/"++nInds++"/"++filt++".lol"
    fileHere <- doesFileExist path
    if fileHere
        then do
            conts <- fmap (map read.lines) (readFile path)
            if length conts >= read nToRet
```

```
                then return (take (read nToRet) conts)
                else do
                    gen <- newStdGen
                    let nToGen = read nToRet - length conts
                    let newConts = genLOLs nToGen (read nPrps)
                                           (read nInds) filt gen
                    appendFile path (ums newConts)
                    return (conts ++ newConts)
        else do
            createDirectoryIfMissing True
                                       ("data/"++nPrps++"/"++nInds)
            gen <- newStdGen
            let conts = genLOLs (read nToRet) (read nPrps)
                                (read nInds) filt gen
            writeFile path (ums conts)
            return conts
```

The final underlying data is belief profiles.

int.hs

```
genBELs :: Int -> Int -> Int -> (Float,Float) -> StdGen
                                                    -> [Profile BEL]
genBELs nToGen nPrps nInds (w1,w2) =
    take nToGen.map (M.fromAscList.zip [0..])
    .unConcat nInds.rndBELs nPrps w1 w2

returnBELs :: [String] -> IO [Profile BEL]
returnBELs [nToRet, nPrps, nInds, wtPair] = do
    let path = "data/"++nPrps++"/"++nInds++"/"++wtPair++".bel"
    fileHere <- doesFileExist path
    if fileHere
        then do
            conts <- fmap (map read.lines) (readFile path)
            if length conts >= read nToRet
                then return (take (read nToRet) conts)
                else do
                    gen <- newStdGen
                    let nToGen = read nToRet - length conts
                    let newConts = genBELs nToGen (read nPrps)
                                           (read nInds) (read wtPair)
                                           gen
                    appendFile path (ums newConts)
                    return (conts ++ newConts)
        else do
            createDirectoryIfMissing True
                                       ("data/"++nPrps++"/"++nInds)
            gen <- newStdGen
            let conts = genBELs (read nToRet) (read nPrps)
                                (read nInds) (read wtPair) gen
            writeFile path (ums conts)
            return conts
```

We use these in various ways. For instance, generating the 'true' preference profiles over actions, taking 0 as the 'real' world, as per Section 4.6. Note here

there is no possibility to simply append to the file: either it is large enough or it gets completely rewritten.

int.hs

```
returnTRUs :: [String] -> IO [Profile (LOL' Action)]
returnTRUs [nToRet,nPrps,nActs,nInds,filt] = do
    let path = "data/"++nPrps++"/"++nInds++"/"++
                nActs++".tsq."++filt++".lol."++".tru"
    fileHere <- doesFileExist path
    if fileHere
        then do
            conts <- fmap (map read.lines) (readFile path)
            if length conts >= read nToRet
                then return (take (read nToRet) conts)
                else do
                    tsqs <- returnTSQs [nToRet,nPrps,nActs]
                    lols <- returnLOLs [nToRet,nPrps,nInds,filt]
                    let newConts = zipWith realActProf tsqs lols
                    writeFile path (ums newConts)
                    return newConts
        else do
            tsqs <- returnTSQs [nToRet,nPrps,nActs]
            lols <- returnLOLs [nToRet,nPrps,nInds,filt]
            let newConts = zipWith realActProf tsqs lols
            writeFile path (ums newConts)
            return newConts
```

We also write the outcome of rules to files.

int.hs

```
returnRULs :: [String] -> IO [LOL' Action]
returnRULs [nToRet,nPrps,nActs,nInds,filt,wtP,rS] = do
    let path = "data/"++nPrps++"/"++nInds++"/"++
                nActs++".tsq."++filt++".lol."++wtP++".bel"
                ++rS++".rul"
    fileHere <- doesFileExist path
    if fileHere
        then do
            conts <- fmap (map read.lines) (readFile path)
            if length conts >= read nToRet
                then return (take (read nToRet) conts)
                else do
                    tsqs <- returnTSQs [nToRet,nPrps,nActs]
                    lols <- returnLOLs [nToRet,nPrps,nInds,filt]
                    bels <- returnBELs [nToRet,nPrps,nInds,wtP]
                    let r = (fromJust.lookup rS) listRules
                    let newConts = zipWith3 r tsqs lols bels
                    writeFile path (ums newConts)
                    return newConts
        else do
            tsqs <- returnTSQs [nToRet,nPrps,nActs]
            lols <- returnLOLs [nToRet,nPrps,nInds,filt]
            bels <- returnBELs [nToRet,nPrps,nInds,wtP]
            let r = (fromJust.lookup rS) listRules
```

```
            let newConts = zipWith3 r tsqs lols bels
            writeFile path (ums newConts)
            return newConts
```

Now we can combine truthful data with the outcomes of two rules to get comparison data.

int.hs

```
returnCMPs :: [String] -> IO [Profile Ordering]
returnCMPs [nRt,nPs,nAs,nIs,flt,wp,r1,r2] = do
    let path = "data/"++nPs++"/"++nIs++"/"++
               nAs++".tsq."++flt++".lol."++wp++".bel"
               ++r1++".rul"++r2++".rul.cmp"
    fileHere <- doesFileExist path
    if fileHere
        then do
            conts <- fmap (map read.lines) (readFile path)
            if length conts >= read nRt
                then return (take (read nRt) conts)
                else do
                    tru <- returnTRUs [nRt,nPs,nAs,nIs,flt]
                    r1o <- returnRULs [nRt,nPs,nAs,nIs,flt,wp,r1]
                    r2o <- returnRULs [nRt,nPs,nAs,nIs,flt,wp,r2]
                    let newConts = zipWith3 cmpActProf r1o r2o tru
                    writeFile path (ums newConts)
                    return newConts
        else do
            tru <- returnTRUs [nRt,nPs,nAs,nIs,flt]
            r1o <- returnRULs [nRt,nPs,nAs,nIs,flt,wp,r1]
            r2o <- returnRULs [nRt,nPs,nAs,nIs,flt,wp,r2]
            let cont = zipWith3 cmpActProf r1o r2o tru
            writeFile path (ums cont)
            return cont
```

Finally, what we want is to return the pairs as per `countOrd`.

int.hs

```
countPref :: Profile Ordering -> (Int,Int)
countPref = M.foldr h1 (0,0)
    where h1 GT (x,y) = (x+1,y)
          h1 LT (x,y) = (x,y+1)
          h1 EQ (x,y) = (x,y)

addPair :: (Num a, Num b) => (a,b) -> (a,b) -> (a,b)
addPair (x,y) (z,w) = (x+z,y+w)

addMaj :: (Ord a,Num a) => (a,a) -> (a,a) -> (a,a)
addMaj (x,y) (z,w) | z > w = (x+1,y)
                   | w > z = (x,y+1)
                   | otherwise = (x,y)

countOrd :: [Profile Ordering] -> ((Int,Int),(Int,Int))
countOrd prf = let prs = map countPref prf
               in  (foldl1 addPair prs,foldl1 addMaj prs)
```

```
showCmp :: [String] -> IO ((Int,Int),(Int,Int))
showCmp xs = do
    ords <- returnCMPs xs
    return (countOrd ords)
```

B.2.4 Specific test data

The first test we ran was that of the decisiveness of rules.

int.hs

```
tWts = ["(0.2,0.1)","(0.5,0.1)","(0.5,0.4)","(0.8,0.1)"]
tPrp = ["1","2","3"]
tAct = ["0","2"]
tInd = ["2","5","10"]
data1 :: [String] -> IO ()
data1 _ = do
    writeFile "data/data1.txt" ""
    forM_ ["earl_bord_approval_pes"] (\rul-> do
        forM_ tPrp (\nP-> do
            forM_ tAct (\nA-> do
                forM_ tInd (\nI-> do
                    forM_ tWts (\wt-> do
                    outcomeList <- returnRULs ["10000",nP,nA,nI
                                               ,"none",wt,rul]
                    let result = countDecisive outcomeList
                    putStrLn $ nP++"-"++nA++"-"++nI++"-"++wt++"-"
                                ++rul++":␣"++show result
                    appendFile "data/data1.txt" $ nP++"␣a␣"++nA
                                    ++"␣i␣"++nI++"␣"++wt++"␣"++
                                    rul++"␣"++show result++"\n"
                        )
                    )
                )
            )
        )
```

Having used this, the next test the decisiveness of a few different rules.

int.hs

```
data2 :: [String] -> IO ()
data2 _ = do
    writeFile "data/data2.txt" ""
    forM_ ["earl_cope_approval_pes"
          ,"earl_bord_approval_ind","late_pes_bord"
          ,"late_ind_bord","beli_approval_ind_bord"
          ,"pref_bord_ind_bord","late_ind_dict"
          ] (\rul-> do
        outcomeList <- returnRULs ["10000","2"
                                   ,"2","5","none"
                                   ,"(0.2,0.1)",rul]
        let result = countDecisive outcomeList
        putStrLn $ rul++":␣"++show result
        appendFile "data/data2.txt" $ rul++"␣"++show result++"\n"
        )
```

```
data3 :: [String] -> IO ()
data3 _ = do
    writeFile "data/data3.txt" ""
    forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
          ,"earl_bord_approval_ind","late_pes_bord"
          ,"late_ind_bord","beli_approval_ind_bord"
          ,"pref_bord_ind_bord","late_ind_dict"
          ] (\rul-> do
        outcomeList <- returnRULs ["10000","2"
                                   ,"0","5","none"
                                   ,"(0.2,0.1)",rul]
        let result = countDecisive outcomeList
        putStrLn $ rul++":␣"++show result
        appendFile "data/data3.txt" $ rul++"␣"++show result++"\n"
        )

data4 :: [String] -> IO ()
data4 _ = do
    writeFile "data/data4.txt" ""
    forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
          ,"earl_bord_approval_ind","late_pes_bord"
          ,"late_ind_bord","beli_approval_ind_bord"
          ,"pref_bord_ind_bord","late_ind_dict"
          ] (\rul-> do
        outcomeList <- returnRULs ["10000","3"
                                   ,"0","2","none"
                                   ,"(0.5,0.1)",rul]
        let result = countDecisive outcomeList
        putStrLn $ rul++":␣"++show result
        appendFile "data/data4.txt" $ rul++"␣"++show result++"\n"
        )

data5 :: [String] -> IO ()
data5 _ = do
    writeFile "data/data5.txt" ""
    forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
          ,"earl_bord_approval_ind","late_pes_bord"
          ,"late_ind_bord","beli_approval_ind_bord"
          ,"pref_bord_ind_bord","late_ind_dict"
          ] (\rul-> do
        outcomeList <- returnRULs ["10000","2"
                                   ,"0","10","none"
                                   ,"(0.8,0.1)",rul]
        let result = countDecisive outcomeList
        putStrLn $ rul++":␣"++show result
        appendFile "data/data5.txt" $ rul++"␣"++show result++"\n"
        )

data6 :: [String] -> IO ()
data6 _ = do
    writeFile "data/data6.txt" ""
    forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
          ,"earl_bord_approval_ind","late_pes_bord"
```

```
          ,"late_ind_bord","beli_approval_ind_bord"
          ,"pref_bord_ind_bord","late_ind_dict"
          ] (\rul-> do
        outcomeList <- returnRULs ["10000","2"
                                   ,"0","10","none"
                                   ,"(0.2,0.1)",rul]
        let result = countDecisive outcomeList
        putStrLn $ rul++":␣"++show result
        appendFile "data/data6.txt" $ rul++"␣"++show result++"\n"
        )

data7 :: [String] -> IO ()
data7 _ = do
    writeFile "data/data7.txt" ""
    forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
          ,"earl_bord_approval_ind","late_pes_bord"
          ,"late_ind_bord","beli_approval_ind_bord"
          ,"pref_bord_ind_bord","late_ind_dict"
          ] (\rul-> do
        outcomeList <- returnRULs ["10000","2"
                                   ,"0","2","none"
                                   ,"(0.2,0.1)",rul]
        let result = countDecisive outcomeList
        putStrLn $ rul++":␣"++show result
        appendFile "data/data7.txt" $ rul++"␣"++show result++"\n"
        )

dataD :: [String] -> IO ()
dataD _ = do
    writeFile "data/dataD.txt" ""
    forM_ ["2","5","10"] (\nI-> do
        forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
              ,"earl_bord_approval_ind","late_pes_bord"
              ,"late_ind_bord","beli_approval_ind_bord"
              ,"pref_bord_ind_bord","late_ind_dict"
              ] (\rul-> do
            outcomeList <- returnRULs ["10000","2"
                                       ,"0","2","none"
                                       ,"(0.2,0.1)",rul]
            let result = countDecisive outcomeList
            putStrLn $ nI++"-"++rul++":␣"++show result
            appendFile "data/dataD.txt" $ nI++"␣"++rul++"␣"
                                          ++show result++"\n"
            )
        )

dataD1 :: [String] -> IO ()
dataD1 _ = do
    writeFile "data/dataD1.txt" ""
    forM_ ["2","5","10"] (\nI-> do
        forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
              ,"earl_bord_approval_ind","late_pes_bord"
              ,"late_ind_bord","beli_approval_ind_bord"
              ,"pref_bord_ind_bord","late_ind_dict"
```

```
              ] (\rul-> do
            outcomeList <- returnRULs ["10000","2"
                                      ,"0",nI,"none"
                                      ,"(0.8,0.1)",rul]
            let result = countDecisive outcomeList
            putStrLn $ nI++"-"++rul++":␣"++show result
            appendFile "data/dataD1.txt" $ nI++"␣"++rul++"␣"
                                           ++show result++"\n"
            )
        )

dataD2 :: [String] -> IO ()
dataD2 _ = do
    writeFile "data/dataD2.txt" ""
    forM_ ["2","5","10"] (\nI-> do
        forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
              ,"earl_bord_approval_ind","late_pes_bord"
              ,"late_ind_bord","beli_approval_ind_bord"
              ,"pref_bord_ind_bord","late_ind_dict"
              ] (\rul-> do
            outcomeList <- returnRULs ["10000","3"
                                      ,"0",nI,"none"
                                      ,"(0.2,0.1)",rul]
            let result = countDecisive outcomeList
            putStrLn $ nI++"-"++rul++":␣"++show result
            appendFile "data/dataD2.txt" $ nI++"␣"++rul++"␣"
                                           ++show result++"\n"
            )
        )

dataD3 :: [String] -> IO ()
dataD3 _ = do
    writeFile "data/dataD3.txt" ""
    forM_ ["2","5","10"] (\nI-> do
        forM_ ["earl_bord_approval_pes","earl_cope_approval_pes"
              ,"earl_bord_approval_ind","late_pes_bord"
              ,"late_ind_bord","beli_approval_ind_bord"
              ,"pref_bord_ind_bord","late_ind_dict"
              ] (\rul-> do
            outcomeList <- returnRULs ["10000","3"
                                      ,"0",nI,"none"
                                      ,"(0.8,0.1)",rul]
            let result = countDecisive outcomeList
            putStrLn $ nI++"-"++rul++":␣"++show result
            appendFile "data/dataD3.txt" $ nI++"␣"++rul++"␣"
                                           ++show result++"\n"
            )
        )
```

The above all concerned decisiveness. The following concern comparisons. We first list the rules we will compare, then the pairs of such, then we do the actual comparing in `dataC`.

int.hs

```
testRules = ["earl_bord_approval_ind","late_ind_bord"
```

```
            ,"beli_approval_ind_bord"
            ,"pref_bord_ind_bord"]

dPairs :: [a] -> [(a,a)]
dPairs (x:[]) = []
dPairs (x:xs) = map (\a->(x,a)) xs ++ dPairs xs

dataC :: [String] -> IO ()
dataC _ = do
    writeFile "data/dataC.txt" ""
    forM_ ["2","5"] (\nI-> do
        forM_ ["1","2"] (\nP-> do
            forM_ ["(0.2,0.1)","(0.5,0.1)"] (\wt-> do
                forM_ (dPairs testRules) (\(r1,r2)-> do
                    result <- showCmp ["10000",nP,"0",nI
                                       ,"none",wt,r1,r2]
                    putStrLn $ nP++"-"++nI++"-"++wt++
                               "-"++r1++"-"++r2++":␣"++show result
                    appendFile "data/dataC.txt" $  nP++"␣"
                                    ++nI++"␣"++wt++"␣"++r1++"␣"
                                    ++r2++"␣"++show result++"\n"
                    )
                )
            )
        )
```

The following helps to filter out strings from a list in a file.

int.hs

```
filterFile :: [String] -> IO String
filterFile (path:xs) = do
    fmap (unlines.filters xs.lines) (readFile path)
        where filters (str:xs) list = filters xs
                                        (filter (isInfixOf str) list)
              filters []       list = list

ioShow :: IO String -> IO ()
ioShow iostr = do
    str <- iostr
    putStrLn str
```

B.2.5 The Kemeny distance

This section considers another way of comparing the outcomes of two actions. Instead of simply comparing the top action suggested by two rules, we could compare how close the full ranking is to what the actual ranking should be, for each individual. One method for measuring the distance between rankings is the Kemeny distance (1959) (cf. to the Kendall tau distance (1938)). This is best implemented via relations.

Agg.hs

```
type REL = S.Set (Int,Int)
```

To apply this fully we must first translate lists of lists to the relation data type. The helper `maxLOL` finds the largest (numerically) state in a list of lists. We assume that all numbers smaller than this are contained in the list of lists.

Agg/Rules.hs

```
ordToREL :: Int -> ORD -> REL
ordToREL n cmp =
    let allPairs       = S.fromAscList
                             [(x,y) | x<-[0..n], y<-[0..n]]
        prefPair (x,y) = x`cmp`y /= LT
    in  S.filter prefPair allPairs

maxLOL :: LOL -> Int
maxLOL = foldl' (flip (max.maximum)) 0

lolToREL :: LOL -> REL
lolToREL xs = ordToREL (maxLOL xs) $ lolToORD xs

kemenyDistanceREL :: REL -> REL -> Int
kemenyDistanceREL r1 r2 = S.size (S.difference r1 r2) +
                          S.size (S.difference r1 r2)

kemenyDistance :: LOL -> LOL -> Int
kemenyDistance xs ys = kemenyDistanceREL (lolToREL xs)
                                         (lolToREL ys)
```

B.2.6 Single peaked preferences

A property that rankings may or may not have is *single-peakedness* (see, for example, work by Austen-Smith and Banks (2000)). The following defines a function that discerns this property.

The function `addIncrsing` takes adjacent elements in a list, and inserts them into some other set of pairs: it adds the increasing pairs of a list. Its inverse is `addDecrsing`, which does the same, only with adjacent pairs taken as decreasing. Utilising both these, `peakREL` creates a set of relations that describes a peak, given the peak element and list. That is, it gives the required pairs for a relation to have a specific peak. The function `vallREL` does the same but for a valley.

Single peakedness is tested relative to some single dimension, i.e. some list in `[Int]`. We have a given preferences, which we represent as a set of pairs, that we want to test is single peaked. The peak is the most preferred element in this set. We must check whether the given set of pairs is a superset of the peak relation of `peakREL` created by the peak and the list. Conversely, for a single-plateau, we require an empty intersection with the `valREL` function's output.

Agg/Rules.hs

```
addIncrsing (x:y:ys) set = S.insert (x,y) $
                                    addIncrsing (y:ys) set
addIncrsing _        set = set

addDecrsing :: [Int] -> REL -> REL
addDecrsing (x:y:ys) set = S.insert (y,x) $
```

```
                                    addDecrsing (y:ys) set
addDecrsing _        set = set

peakREL :: Int -> [Int] -> REL
peakREL p xs = let (ups,downs) = break (==p) xs
               in  addIncrsing ups
                               (addDecrsing downs S.empty)

vallREL :: Int -> [Int] -> REL
vallREL p xs = let (downs,ups) = break (==p) xs
               in  addIncrsing ups
                               (addDecrsing downs S.empty)

isSinglePeak :: [Int] -> LOL -> Bool
isSinglePeak xs lol
    | hasUniWin lol = peakREL ((head.head) lol) xs
                                        `S.isSubsetOf`
                                                lolToREL lol
    | otherwise     = False

isSinglePlat :: [Int] -> LOL -> Bool
isSinglePlat xs lol
    | hasUniWin lol =
                  let vals = vallREL ((head.head) lol)
                                     xs
                      rels = lolToREL lol
                  in  rels `S.intersection` vals
                                          == S.empty
    | otherwise     = False
```

More generally, we may want to be able to filter preferences that are extensions of some other ordering, `extendsPref`, or those that do not contain a specific order pair, `inadmissiblePair`.

Agg/Rules.hs

```
                       -- [[Int]] may be "incomplete"
extendsPref lol cond = lolToREL cond `S.isSubsetOf`
                                                lolToREL lol


inadmissiblePair :: LOL -> (Int,Int) -> Bool
inadmissiblePair lol pair = not $
                            S.member pair $ lolToREL lol
```

B.3 Miscellany

B.3.1 Ostrogorski paradox

The following statement of Ostrogorski's paradox was given originally by Kelly (1989). There are two candidates, α and β, and three political issues, p, q and r on which the candidates have taken positions. There are five voters; each will vote for the candidate with whom he agrees on more issues. The position and preference data are given in Table B.1.

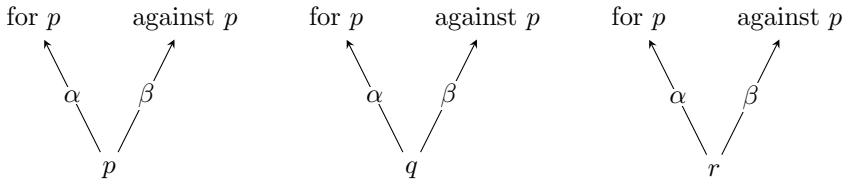
Table B.1: Ostrogorski's paradox.

	Issue		
	p	q	r
Candidate			
α	for	for	for
β	against	against	against
Voter #			
1	for	for	against
2	for	against	for
3	against	for	for
4	against	against	against
5	against	against	against

We can model the issues as input states, the candidates as actions, and the positions as output states, as in Figure B.1. The Ostrogorski paradox then mirrors the choice between early- and late-aggregation.

Figure B.1 Ostrogorski's paradox as a setup.

Transition function:



Preferences: for example, voter one has preferences:

$$\text{for } p \rhd \text{against } p \qquad \text{for } q \rhd \text{against } q \qquad \text{against } r \rhd \text{for } r$$

Note this is not a weak order.

Bibliography

Jorge Alcalde-Unzu and Marc Vorsatz. Size approval voting. *Journal of Economic Theory*, 144(3):1187–1210, 2009. 60

Armen A Alchian. The meaning of utility measurement. *The American Economic Review*, pages 26–50, 1953. 47

Maurice Allais. Le comportement de l'homme rationnel devant le risque: critique des postulats et axiomes de l'école américaine. *Econometrica: Journal of the Econometric Society*, pages 503–546, 1953. 46

Rixtar Arlegi. A note on Bossert, Pattanaik and Xu's" choice under complete uncertainty: Axiomatic characterization of some decision rules. *Documentos de Trabajo (Universidad Pública de Navarra. Departamento de Economía)*, (2):1, 2002. 25, 51

Kenneth Arrow, Sen Amartya, and Kotaro Suzumura. *Handbook of social choice and welfare*. Amsterdam:. North-Holland,, 2002. 6

Kenneth J Arrow. Social choice and individual values. *Monograph/Cowles Foundation for Research in Economics at Yale University*, (12), 1963. 14, 16, 17

David Austen-Smith and Jeffrey S Banks. *Positive political theory I: collective preference*, volume 1. University of Michigan Press, 2000. 98

Salvador Barbera, Walter Bossert, and Prasanta K Pattanaik. *Ranking sets of objects*. Springer, 2004. 25, 31, 51

Charles Blackorby, David Donaldson, and John A Weymark. Social choice with interpersonal utility comparisons: a diagrammatic introduction. *International Economic Review*, pages 327–356, 1984. 50

Julian H Blau. Arrow's theorem with weak independence. *Economica*, pages 413–420, 1971. 16

Jean C de Borda. Mémoire sur les élections au scrutin. 1781. 11, 61

Walter Bossert and Arkadii Slinko. Relative uncertainty aversion and additively representable set rankings. *International Journal of Economic Theory*, 2(2): 105–122, 2006. 52

Walter Bossert and John A Weymark. Utility in social choice. In *Handbook of utility theory*, pages 1099–1177. Springer, 2004. 50

Walter Bossert, Prasanta K Pattanaik, and Yongsheng Xu. Choice under complete uncertainty: axiomatic characterizations of some decision rules. *Economic Theory*, 16(2):295–312, 2000. 51

Craig Boutilier. Toward a logic for qualitative decision theory. *KR*, 94:75–86, 1994. 48

Ronen I Brafman and Moshe Tennenholtz. On the foundations of qualitative decision theory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1291–1296, 1996. 48

Steven J Brams and Peter C Fishburn. *Approval voting*. Springer, 2007. 59

Steven J Brams, D Marc Kilgour, and William S Zwicker. The paradox of multiple elections. *Social Choice and Welfare*, 15(2):211–236, 1998. 54

Stanley Burris and HP Sankappanavar. *A course in universal algebra*. New York, 1981. 32

Marston Conder, Simon Marshall, and Arkadii Slinko. Rankings of multisets and discrete cones. Technical report, Department of Mathematics, The University of Auckland, New Zealand, 2004. 53

Marston Conder, Simon Marshall, and Arkadii Slinko. Orders on multisets and discrete cones. *Order*, 24(4):277–296, 2007. 29, 31, 53

Marquis de Condorcet. Essai sur lapplication de lanalyse à la probabilité de décisions rendues à la pluralité de voix, imprimerie royal, 1785. 15, 61

Claude d'Aspremont and Louis Gevers. Social welfare functionals and interpersonal comparability. *Handbook of social choice and welfare*, 1:459–541, 2002. 50, 51

Franz Dietrich and Christian List. The aggregation of propositional attitudes: towards a general theory. 2008. 53

Kees Doets and Jan van Eijck. *The Haskell Road to Logic, Maths and Programming (Texts in Computing)*. College Publications, 2004. 55

Jon Doyle and Michael P Wellman. Impediments to universal preference-based default theories. *Artificial Intelligence*, 49(1):97–128, 1991. 53

Didier Dubois and Henri Prade. Possibility theory as a basis for qualitative decision theory. In *IJCAI*, volume 95, pages 1924–1930, 1995. 48

Didier Dubois, Hélène Fargier, and Henri Prade. Decision-making under ordinal preferences and comparative uncertainty. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 157–164. Morgan Kaufmann Publishers Inc., 1997. 48, 63

Didier Dubois, Hélène Fargier, Henri Prade, and Patrice Perny. Qualitative decision theory: from Savage's axioms to nonmonotonic reasoning. *Journal of the ACM (JACM)*, 49(4):455–495, 2002. 49

Didier Dubois, Hélène Fargier, and Patrice Perny. Qualitative decision theory with preference relations and comparative uncertainty: An axiomatic approach. *Artificial Intelligence*, 148(1):219–260, 2003. 6, 49

Conal Duddy and Ashley Piggins. Collective approval. *Mathematical Social Sciences*, 65(3):190–194, 2013. 59

Bhaskar Dutta and Arunava Sen. Ranking opportunity sets and arrow impossibility theorems: correspondence results. *Journal of Economic Theory*, 71(1): 90–101, 1996. 51

Daniel Ellsberg. Risk, ambiguity, and the Savage axioms. *The quarterly journal of economics*, pages 643–669, 1961. 46

Ulle Endriss. Voting on actions with uncertain outcomes. In *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT-2013)*, volume 8176 of *LNAI*, pages 167–180. Springer-Verlag, 2013. 6, 15, 24, 25, 39, 42, 44, 45, 51, 53, 55, 57, 59, 60

Hélene Fargier and Patrice Perny. Qualitative models for decision under uncertainty without the commensurability assumption. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 188–195. Morgan Kaufmann Publishers Inc., 1999. 48

Peter C Fishburn and William V Gehrlein. Borda's rule, positional voting, and condorcet's simple majority principle. *Public Choice*, 28(1):79–88, 1976. 11, 61

Wulf Gaertner. *A Primer in Social Choice Theory: Revised Edition: Revised Edition*. Oxford University Press, 2009. 6

John Geanakoplos. Three brief proofs of Arrows impossibility theorem. *Economic Theory*, 26(1):211–215, 2005. 17

KP Girish and Jacob John Sunil. General relations between partially ordered multisets and their chains and antichains. *Mathematical Communications*, 14 (2):193–205, 2009. 25

Dominique Henriet. The Copeland choice function: an axiomatic characterization. *Social Choice and Welfare*, 2(1):49–63, 1985. 61

OEIS Foundation Inc. The on-line encyclopedia of integer sequences. `http://oeis.org`, 2011. Accessed June 25, 2014. 12

Richard Jeffrey. The sure thing principle. In *PSA: Proceedings of the biennial meeting of the philosophy of science association*, pages 719–730. JSTOR, 1982. 45, 46, 47

Yakar Kannai and Bezalel Peleg. A note on the extension of an order on a set to the power set. *Journal of Economic Theory*, 32(1):172–175, 1984. 52

Jerry S Kelly. The Ostrogorski paradox. *Social Choice and Welfare*, 6(1):71–76, 1989. 99

John G Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959. 97

Maurice G Kendall. A new measure of rank correlation. *Biometrika*, pages 81–93, 1938. 97

Donald E Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984. 55, 83

M. Kochanski. How many orders are there? `http://www.nugae.com/mathematics/bin/ordering.pdf`, 2007. Accessed June 25, 2014. 70

Tjalling Charles Koopmans. *On flexibility of future preference*. Cowles Foundation for Research in Economics at Yale University, 1964. 51

David M Kreps. A representation theorem for "preference for flexibility". *Econometrica: Journal of the Econometric Society*, pages 565–577, 1979. 51

Christian List. The theory of judgment aggregation: An introductory review. *Synthese*, 187(1):179–207, 2012. 53

Harry M Markowitz. *Portfolio Selection: Efficient Diversification of Investments*, volume 16. Yale University Press, 1959.

Ursula Martin. A geometrical approach to multiset orderings. *Theoretical Computer Science*, 67(1):37–54, 1989. 53

Kenneth O May. A set of independent necessary and sufficient conditions for simple majority decision. *Econometrica: Journal of the Econometric Society*, pages 680–684, 1952. 20

Vilfredo Pareto. On the equilibrium of the social system. In *Theories of society: foundations of modern sociological theory*, pages 1288–1292. Free Press of Glencoe, Inc., 1935. 15

Marcus Pivato. Social welfare with incomplete ordinal interpersonal comparisons. *Journal of Mathematical Economics*, 49(5):405–417, 2013. 53

Jeffrey Richelson. Conditions on social choice functions. *Public Choice*, 31(1): 79–110, 1977. 14

John E Roemer. *Theories of distributive justice*. Harvard University Press, 1998. 49

Dov Samet. The sure-thing principle and independence of irrelevant knowledge. 2008. 48

Leonard J Savage. *The foundations of statistics*. Courier Dover Publications, 1972. 6, 7, 43

Leonard J Savage et al. *The foundations of statistics reconsidered*. University of California Press, 1961. 43

Amartya K Sen. *Collective choice and social welfare*. North-Holland Publishing Co., 1970. 49

Amartya K Sen. Choice functions and revealed preference. *The Review of Economic Studies*, pages 307–317, 1971. 51

Murat Sertel and Arkadii Slinko. Ranking committees, words or multisets. Technical report, Department of Mathematics, The University of Auckland, New Zealand, 2002. 53

Lloyd S Shapley. Cores of convex games. *International journal of game theory*, 1(1):11–26, 1971. 54

Arkadii Slinko. Additive representability of finite measurement structures. In *The Mathematics of Preference, Choice and Order*, pages 113–133. Springer, 2009. 53

Wolfgang Spohn. A general non-probabilistic theory of inductive reasoning. *arXiv preprint arXiv:1304.2375*, 2013. 53

Sek-Wah Tan and Judea Pearl. Qualitative decision theory. In *AAAI*, pages 928–933, 1994. 48

T Nicolaus Tideman and Florenz Plassmann. Modeling the outcomes of vote-casting in actual elections. In *Electoral Systems*, pages 217–251. Springer, 2012. 55

Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Pieter Kooi. *Dynamic epistemic logic*, volume 337. Springer, 2007. 53

Abraham Wald. Contributions to the theory of statistical estimation and testing hypotheses. *The Annals of Mathematical Statistics*, 10(4):299–326, 1939. 6, 43

Martin Weber. Decision making with incomplete information. *European Journal of Operational Research*, 28(1):44–57, 1987. 53

Yongsheng Xu. On ranking compact and comprehensive opportunity sets. *Mathematical Social Sciences*, 45(2):109–119, 2003. 51

Yongsheng Xu. Axiomatizations of approval voting. In *Handbook on Approval Voting*, pages 91–102. Springer, 2010. 60

H Peyton Young. An axiomatization of borda's rule. *Journal of Economic Theory*, 9(1):43–52, 1974. 61