CDMTCS
Research
Report
Series

Universal Semimeasures: An Introduction

Nicholas J. Hay
University of Auckland, NZ

CDMTCS-300
February 2007

Centre for Discrete Mathematics and
Theoretical Computer Science

Universal Semimeasures: An Introduction

Nicholas James Hay

under the supervision of

Professor Cristian S. Calude and Dr. André Nies

A thesis submitted in fulfilment of the requirements
for the degree of Master of Science in Computer Science,
The University of Auckland, 2007.

Abstract

Universal semimeasures [Hut05][LV97], a type of function derived from probability and computability theory, describe extremely powerful sequence predictors. They outperform all other predictors but for a penalty no more than the predictor's complexity. They learn to predict any computable sequence with error no more than the sequence's complexity. Universal semimeasures work by modelling the sequence as generated by an unknown program running on a universal computer. Although these predictors are uncomputable, and so cannot be implemented in practice, they serve to describe an ideal: an existence proof for systems that predict better than humans.

We review the mathematics behind universal semimeasures and discuss some of its implications. Our approach differs from previous ones in several respects. We show semimeasures correspond to probability measures over the set of finite and infinite sequences, establishing a rigorous footing. We demonstrate the existence of universal semimeasures using a novel proof of the equivalence between enumerable semimeasures and processes. We take into account the possibility of sequence termination leading to a stronger definition of universality. To make explicit hidden constants we define a novel complexity measure, simulation complexity, which generalises monotone complexity. Finally, we emphasise the use of logarithmic scoring rules [Ber79] to measure error in prediction.

Acknowledgements

First, thanks to my supervisors, Cristian Calude and André Nies, for their comments and advice.

Next, Marcus Hutter and Shane Legg, for their comments on early results presented at the 2006 Dagstuhl Seminar on Kolmogorov Complexity. Alex Raichev, for help with both LaTeX and math. Ludwig Staiger, for his help with measure theory. Fuad Tabba, for general help with writing the thesis. Eliezer Yudkowsky, for his comments on an earlier version of this work, and for discussions which formed the seed for some of the more “philosophical” sections.

Finally, my family for proofreading the less technical parts of the thesis, and for their ever-present support.

Contents

List of Notation

$\mathrm{Err}[p]$	The error of a prediction; equal to $-\log_2 p$	12
$\lvert x\rvert$	The length of a sequence $x \in X^{\#}$	13
$\mathbb{B}$	The set of binary digits; equal to $\{0, 1\}$	13
X^*	The set of finite sequences (strings) over the set X	14
X^∞	The set of infinite sequences over the set X	14
$X^{\#}$	The set of finite and infinite sequences over the set X	14
ϵ	The empty string	14
xy	The concatenation of two sequences x and y	14
$x \preceq y$	The sequence x is a prefix of the sequence y	14
$x_{i:j}$	The substring from the ith to jth character	14
$\sup S$	The shortest sequence extending every element of S	14
ν, ρ	Typical semimeasure variables	16
Γ_x	A cylinder set, the set of sequences extending x	18
Σ	The smallest σ-algebra containing all cylinder sets	19
Σ_0	The smallest algebra containing all cylinder sets	20
λ	The uniform semimeasure	40
μ_M	The semimeasure generated by process M	41
$\mathcal{M}\,[r]$	The process generated by request sequence r	48
$S_U(M)$	The weak simulation complexity of the process M relative to U	88
$S_U(M\!\downarrow)$	The strong simulation complexity of the process M relative to U	88
$S_U(\nu)$	The weak simulation complexity of the semimeasure ν	89
$S_U(\nu\!\downarrow)$	The strong simulation complexity of the semimeasure ν	89
$S_U(x)$	The weak simulation complexity of the sequence x	89
$S_U(x\!\downarrow)$	The strong simulation complexity of the sequence x	89
Γ_x^0	The set of *infinite* sequences extending x	99

Introduction

Suppose you are predicting next week's weather in order to decide whether to cancel the county fair. This prediction can take into account a wide variety of information: the weather today, the newspaper's forecast, whether your knee is aching, the temperature distribution over the Pacific, or numerical simulations of the atmosphere. This information is distilled into whether it will be rainy, sunny, cloudy, or snowy on each day. In general there is uncertainty about which sequence of outcomes will occur so we use probabilities, numbers between 0 and 1, to weigh how likely each possibility is [Jay03] (see Section 1.1 for an example and further details).

Suppose instead you are trying to predict the output of a computer running a completely unknown (or random) program. A computer program is a list of instructions used to generate a sequence. For example, "output twelve 0's" is such a program. Programs are encoded into a sequence of bits, for instance the previous program might correspond to the sequence "00001100110". If we know nothing about the program each bit is equally likely to be 0 or 1. The probability of any particular program of length n is then $1/2^n$. The probability the computer outputs any particular sequence is the sum of the probabilities of all the programs that output it.

This thesis demonstrates that modelling a sequence *as if* it were generated by a computer running a random program is a generally excellent method for predicting a sequence. These predictors correspond to mathematical objects termed universal semimeasures. This method works because sequences with patterns in them will be generated by programs shorter than themselves. Even if the patterns are subtle, when the sequence is sufficiently long[1] these predictors will perform well. Unfortunately, because some programs never halt these predictors cannot be implemented

[1]Where sufficiently long is more like 2^{20} bits than $2^{2^{20}}$ bits i.e. the constants aren't pathological. Although the sequence of next week's weather (rainy, sunny, cloudy, or snowy) is too short, the sequence of temperature and pressure readings over a country probably isn't.

on a computer[2]. It, however, serves as an ideal to approximate, and the theory we develop also covers predictors we *can* implement.

Chapter 1 gives an informal introduction to probabilities and how they can be used to make decisions. This is applied to derive a measurement of prediction error. Chapter 4 will use this measure to evaluate the performance of universal semimeasures.

Chapter 2 introduces semimeasures, our mathematical tool for representing predictions. These are proven equivalent to probability measures over the set of all finite and infinite sequences. This equivalence is interesting as rigorous probability theory is based on probability measures[3] [Ash72]. As a result, it features in later definitions and proofs.

Chapter 3 adds computability to the mix. It first defines processes, computable systems which receive input and produce output. It next defines enumerable semimeasures, those semimeasures which can be computed from below. These two notions of computability are then proven equivalent. All later results depend on this equivalence.

Chapter 4 constructs universal semimeasures and proves bounds on their performance. It introduces and links the concepts of dominance between semimeasures and simulations between processes. It defines a novel complexity measure, simulation complexity, which features in the bounds proved. It concludes with some remarks why these semimeasures may be highly effective at predicting sequences in our universe, were it possible to implement them.

Connection to literature

This thesis is a self-contained exposition on universal (enumerable) semimeasures, but it does not exist in isolation. Our results are similar those in [Sol64] [ZL70] [Sol78] [LV97] [Leg97] [Hut05]. In the following we compare our approach those

[2]We cannot in general separate programs that halt from those which don't (this is the halting problem [Odi89]).

[3]See, however, [Jay03] for cautionary remarks on why measures over infinite sets should be derived from those over finite sets as a limit. This can be done in our case by treating $X^{\#}$, the set of finite and infinite sequences, as the limit of $X^{\leq n}$, the set of sequences with length at most n. Jaynes also offers an illuminating derivation of probability theory as classical logic extended to uncertain propositions, and a full treatment of the theory from this perspective.

before us. This will be interesting to readers who are familiar with the field, or who have already read the bulk of the thesis. For history and further references see [Hut05] and [LV97].

1. We differ from [Hut05] and [LV97] in our treatment of semimeasures. As Section 2.3 demonstrates, semimeasures are best seen as actual measures over the set of finite and infinite sequences $X^{\#}$ than defective measures over the set of infinite sequences X^{∞}. This is why we insist on $\nu(\epsilon) = 1$ rather than the weaker $\nu(\epsilon) \leq 1$ in Definition 2.4.

2. Following the above point, in general we allow the possibility of finite sequences. This leads to a stronger definition of dominance and universality (see Definitions 4.1 and 4.13, and Section 4.1.1). Strong dominance takes into account the semimeasure's performance on finite as well as infinite sequences. The standard universal semimeasure constructions, either as mixtures of all enumerable semimeasures or as the semimeasure of a universal process, satisfy this stronger definition.

3. In algorithmic information theory [Cal02] [LV97] bounds are typically proven up to an unspecified additive constant. For our results the size of constants are important, the smaller the better, so all constants are labelled. These constants turn out to have natural interpretations. This leads us to define a new form of complexity, termed simulation complexity (Section 4.2.2), which generalises monotone complexity.

4. Strictly proper scoring rules [GR04] are natural ways to measure the error of a prediction[4]. For example, Solomonoff's original result [Sol78] was posed in terms of the quadratic score. Following [Ber79] we argue the logarithmic scoring rule is the appropriate way to measure prediction error. See Section 1.2, Definition 4.1, and Theorem 4.8.

5. For various results it is necessary to define a computable sequence transformer: something which reads a sequence and outputs a sequence, both potentially infinite. This is often defined through monotone Turing machines [LV97] [Hut05]. We define the abstract notion of a process (following [Cal02], and similar to [ZL70]; see Definition 3.6) which axiomatically characterises the

[4]Actually, these measure the accuracy of a prediction. To measure the error one simply negates the measure.

necessary computability properties. We then prove it equivalent to monotone machines (see Section 3.2.2).

To summarise the above, this thesis presents the following novelties:

1. Strong dominance and universality (Definitions 4.1 and 4.13, and Section 4.1.1).
2. Simulation complexity (Section 4.2.2).

It also contains detailed proofs which may be of independent interest:

1. Theorem 2.12: Every semimeasure corresponds to a unique probability measure over the set of finite and infinite sequences $X^{\#}$, and vice versa.
2. Theorem 3.7: Every process is implementable by an interactive algorithm (e.g. a monotone machine), and vice versa.
3. Theorem 3.29: Every enumerable semimeasure is the output semimeasure of some process, and vice versa.
4. Theorem 4.8: If a process simulates another process then its semimeasure dominates the other's.

Chapter 1

Probability theory

Probability theory [Ash72] is a model of reasoning under uncertainty[1] [Jay03]. Probability distributions capture uncertain states of knowledge by consistently assigning probabilities to propositions. For instance, to capture knowledge of a bus schedule we assign probabilities to the propositions "the bus arrives at time t" for each possible arrival time t. As prediction requires the handling of uncertain knowledge, we find predictions are well modelled by probability distributions.

Section 1.1 informally introduces probability theory through its primary application: decision making under uncertainty [Ber93]. Each action will assign probabilities to the possible outcomes that could result from it. Section 1.2 uses the theory of decision making from Section 1.1.1 derive a measure of a prediction's error. This measure will be used throughout Chapter 4.

1.1 Making decisions

A common approach to decision making under uncertainty is expected utility theory, where probabilities are used to capture the predicted effect of each action, and utility functions are used to judge them. We introduce this theory by example.

Consider a racing driver deciding on a strategy. This is a small race, involving only three cars. There are three possible outcomes for a driver: 1st, 2nd, or 3rd place.

[1]Although human reasoning often departs from the predictions of probability theory, there are good reasons to suggest it is human reasoning at fault [KST82] [Jay03]. Regardless, it is a commonly used and particularly elegant model of uncertainty.

There are three possible strategies. First is a safe strategy which can guarantee 2nd place. Second is a risky strategy involving dodging in front of the opponents' cars, giving a higher chance of 1st place than the last strategy but an even higher chance of 3rd place. Finally there is a losing strategy which is a guaranteed 3rd place. Which strategy the driver prefers depends on which outcome he prefers. One driver prefers 1st to 2nd to 3rd. Another driver wants only to win: 2nd is as bad as 3rd. The last wants to lose: he's being blackmailed by one of the other drivers.

Each outcome has a probability depending on the strategy chosen, detailed in Table 1.1. In this example each driver has the same knowledge about which outcomes follow which strategies, so each assign the same probabilities. In general, probabilities can differ.

	1st	2nd	3rd
Safe	0.05	0.80	0.15
Risky	0.20	0.04	0.76
Loss	0.01	0.04	0.95

Table 1.1: Probability of outcome given strategy

Each driver assigns utilities to outcomes depending on their preferences, detailed in Table 1.2. These differ between drivers as they want different outcomes.

	1st	2nd	3rd
Normal driver	1.00	0.50	0.00
Win driver	1.00	0.00	0.00
Blackmailed driver	0.00	0.25	1.00

Table 1.2: Utility of outcome assigned by driver

By multiplying a row of probabilities componentwise with a row of utilities we compute the expected utility each driver assigns each strategy. For example, the normal driver assigns to the risky strategy the expected utility

$$0.20 \times 1 + 0.04 \times 0.5 + 0.76 \times 0 = 0.22,$$

which is simply the second row of Table 1.1 times the first row of Table 1.2. Table 1.3 has the expected utilities for our example.

Each driver picks the strategy they assign the highest expected utility to. For

	Safe	Risky	Loss
Normal driver	0.45	0.22	0.03
Win driver	0.05	0.20	0.01
Lose driver	0.35	0.77	0.96

Table 1.3: Expected utility of strategy assigned by driver

example, the normal driver prefers the safe to the risky strategy, and prefers both to the loss strategy. He chooses the safe strategy. Similarly, the win driver chooses the risky strategy, and the blackmailed driver the loss strategy.

1.1.1 Expected utility theory

In general, we have a set O of possible outcomes and a set A of possible actions. Both outcomes and actions are mutually exclusive and exhaustive: exactly one outcome actually occurs and exactly one action is taken. To each action $a \in A$ and outcome $o \in O$ we assign a probability $p(o|a)$ that outcome o follows from action a.

In the above example we have $O = \{1\text{st}, 2\text{nd}, 3\text{rd}\}$ and $A = \{\text{Safe}, \text{Risky}, \text{Loss}\}$. Table 1.1 gives the probabilities $p(o|a)$ shared between all drivers.

Definition 1.1. A **conditional probability distribution** over O given A is a function $p\colon O \times A \to [0, 1]$ such that

$$\sum_{o \in O} p(o|a) = 1 \quad \text{for all } a \in A.$$

□

A **utility function** $u\colon O \to \mathbb{R}$ assigns each outcome a utility. The larger the utility the more that outcome is preferred. Each row of Table 1.2 is a utility function, one for each driver. A utility function u and conditional probability distribution p combine to compute the **expected utility** of each action $a \in A$

$$E_p[u|a] = \sum_{o \in O} p(o|a)u(o).$$

Each entry of Table 1.3 is an expected utility where p is fixed, u varies between driver, and a varies over strategies. This gives us an ordering[2] over actions. When faced

[2]More precisely, a total preordering: a reflexive, transitive, and total relation over A.

with a decision the decision maker selects an action whose probability distribution is greatest in this order.

Two observations will be important for the following section:

1. The sets O and A can be infinite, even uncountably so. However, for a fixed action a the probability $p(o|a)$ will only be nonzero for countably many o.

2. Different utility functions can embody the same preferences. Given a utility function u define another
$$v(o) = \alpha\, u(o) + \beta$$
for arbitrary $\alpha, \beta \in \mathbb{R}$ with α positive. The parameter α scales u; the parameter β translates it. It is easy to see that
$$E_p[u|a_1] \leq E_p[u|a_2] \quad \text{if and only if} \quad E_p[v|a_1] \leq E_p[v|a_2]$$
holds for all $a_1, a_2 \in A$ and all probability distributions $p\colon O \times A \to [0,1]$.

1.2 Measuring a prediction's error

This section applies the expected utility theory of the previous section to derive a measure of prediction error. This approach is inspired by [Ber79] who proves a similar result.

A weather forecaster assigns probabilities to the possible states of weather. In general a forecaster's[3] forecast is a probability distribution. The process of deciding which forecast to make can be modelled within the expected utility framework introduced above (Section 1.1). Denote by X the set of possibilities being forecast e.g. the possible weather states. Denote by
$$D(X) = \{d\colon X \to [0,1] : \sum_x d(x) = 1\}$$
the set of all possible forecasts.

From the forecaster's point of view, an outcome is a pair $\langle d, x\rangle \in D(X) \times X$ of a forecast and a possibility. The forecaster has knowledge of which outcomes are likely

[3] We use "forecaster" and "forecast" here to avoid confusion. Forecasters are exactly predictors, and forecasts are the same as predictions.

to result from which forecasts, this knowledge being represented by a conditional probability distribution $p\colon (D(X) \times X) \times D(X) \to [0,1]$. This distribution is of the form

$$p(d, x|d_0) = [d_0 = d]p_0(x)$$

where $[d_0 = d]$ equals 1 if $d_0 = d$ is true and 0 otherwise, and where $p_0 \in D(X)$ is the forecaster's knowledge of how likely the possibilities $x \in X$ are. It is of this form because the forecaster knows that the forecast she chooses will be the actual forecast, and that which possibility arises does not depend on the forecast made (a natural assumption). Note that p follows Definition 1.1 of conditional probability distributions with $O = D(X) \times X$ and $A = D(X)$.

The forecaster has a utility function

$$u\colon D(X) \times X \to \mathbb{R}$$

over the set of possible outcomes. The expected utility of a forecast $d_0 \in D(X)$ is then

$$\begin{aligned} E_p[u|d_0] &= \sum_{\substack{d \in D(X) \\ x \in X}} p(d, x|d_0)u(d, x) \\ &= \sum_{x \in X} p_0(x)u(d_0, x). \end{aligned}$$

The forecaster selects a forecast $d^* \in D(X)$ which maximises this sum.

Suppose the forecaster wants to make accurate forecasts. This means her utility function u measures the accuracy of the forecast made. Given this we expect several things:

1. The forecast p_0 has maximum expected utility. This is because there is no way to predict reliably better than your current knowledge. Any other distribution has lower expected utility.

2. A distribution assigning more probability to the correct outcome never has lower utility than one which assigns less. Increasing the probability of the correct outcome can never result in a lower utility.

3. The utility function u is continuously differentiable in $D(X)$. This means the utility function varies "smoothly" as we change the probability of outcomes.

These three properties constrain those utility functions u which measure accuracy.

Theorem 1.2. Let $u\colon D(X)\times X \to \mathbb{R}$ be a real function continuously differentiable in its first argument. If

1. For any fixed $p_0 \in D(X)$ the expected utility
$$\sum_x p_0(x)u(q,x)$$
for $q \in D(X)$ is maximised only when $q = p_0$,
2. For all $q_1, q_2 \in D(X)$ and any $x \in X$, if $q_1(x) < q_2(x)$ then
$$u(q_1,x) \leq u(q_2,x),$$

then
$$u(q,x) = A\log q(x) + B(x)$$
for some $A > 0$ and $B\colon X \to \mathbb{R}$. Conversely, functions of the above form satisfy the preceding properties.

Proof. We first show that if $q_1(x) = q_2(x)$ for $q_1, q_2 \in D(X)$ then $u(q_1,x) = u(q_2,x)$, where $x \in X$ is fixed. This will imply
$$u(q,x) = u_0(q(x),x)$$
for some $u_0\colon [0,1]\times X \to \mathbb{R}$. Suppose this doesn't hold. Then without loss of generality there exists $q_1, q_2 \in D(X)$ where $q_1(x) = q_2(x)$ yet
$$u(q_1,x) > u(q_2,x).$$

As u is continuous in $D(X)$, there is an open ball about q_2 such that the above inequality remains true. In particular, we can choose q_2' such that $q_1(x) < q_2'(x)$ and $u(q_1,x) > u(q_2',x)$. This contradicts the hypothesis of this theorem, so the u_0 above exists.

Fix $p_0 \in D(x)$. By the above,
$$\sum_x p_0(x)u(q,x) = \sum_x p_0(x)u_0(q(x),x).$$

We know $q = p_0$ is the unique maxima of this sum if q is constrained to satisfy $\sum_x q(x) = 1$. We can use this to extract information about u_0. By the method of Lagrange multipliers [HH99] any constrained extrema of the above sum is an unconstrained one of the following

$$F[q] \quad = \quad \sum_x p_0(x)u_0(q(x), x) - A\left(\sum_x q(x) - 1\right).$$

where $A \in \mathbb{R}$ is a Lagrange multipler. If q is an extrema of the above then

$$0 = \frac{\partial}{\partial q(\hat{x})} F(q) = p_0(\hat{x}) D_1 u_0(q(\hat{x}), \hat{x}) - A$$

for all $\hat{x} \in X$ where $D_1 u_0$ denotes the partial derivative of u_0 in its first argument, and $\frac{\partial}{\partial q(\hat{x})}$ the operator of partial differentiation with respect to the $\hat{x}$ argument of the function q. Since p_0 is a maximal distribution, by hypothesis, we know $q = p_0$ satisfies the above identity, resulting in the differential equation

$$D_1 u_0(p_0(\hat{x}), \hat{x}) = A/p_0(\hat{x}) \quad \text{for all } \hat{x} \in X.$$

Since the above must hold for all distributions p_0 it must hold for all values of $p_0(\hat{x})$ within the set $[0, 1]$. This equation has the unique solution

$$u_0(p(\hat{x}), x) = A \log p(\hat{x}) + B(x)$$

for some A and a function $B\colon X \to \mathbb{R}$. Were A zero then all distributions would be maxima, contradicting uniqueness of p. Were A negative then p would be the unique *minimum*. So A is positive.

Conversely, suppose

$$u(q, x) = A \log q(x) + B(x)$$

for some $A > 0$ and $B\colon X \to \mathbb{R}$. Clearly u is continuously differentiable, and $q_1(x) < q_2(x)$ implies $u(q_1, x) \leq u(q_2, x)$. It remains to show $q = p_0$ is the unique maximum of

$$\sum_x p_0(x)u(q, x) = A \sum_x p_0(x) \log q(x) + \sum_x p_0(x)B(x).$$

Since $A > 0$ and $\sum_x p_0(x)B(x)$ is constant in q, the maxima of the above expression are exactly the maxima of

$$\sum_x p_0(x) \log q(x).$$

It is an elementary result [CT91] that this is maximised exactly when $q = p_0$. $\square$

The above theorem establishes that if u measures accuracy, we must have

$$u(q, x) = A \log q(x) + B(x).$$

We can restrict this further. A measure of accuracy should be invariant under permutations of X, so the function $B\colon X \to \mathbb{R}$ should be constant. As translations of utility functions cannot change decisions, we choose $B(x) = 0$. As positive rescaling of utility functions cannot change decisions, we choose A to give us the binary logarithm (we will have probabilities of the form 2^{-n} later). So we have

$$u(q, x) = \log_2 q(x)$$

as our measure of accuracy. As error is negative accuracy (i.e. increased error is decreased accuracy and vice versa) this leads us to the definition of the error of a prediction.

Definition 1.3. The **error** of a prediction is

$$\mathrm{Err}[p] = -\log_2 p$$

where p is the probability assigned to the event which actually occurs. $\square$

Chapter 2

Semimeasures

We are often faced with predicting a sequence: successive states of a process, such as the weather; successive outputs from a process, such as bits from computer or parts from a manufactory; or successive inputs to a process, such as an agent's observations over time. Even the process of prediction naturally forms sequences: predicting multiple things at different times creates a sequence of predictions.

This thesis is about sequence prediction so we require a theory of such predictions. Predictions in general are modelled through probability theory, so we develop probability theory for sequences. The previous chapter included a relatively informal introduction to probability theory, primarily as motivation for our measure of error. This chapter describes formally the probability theory we require for later chapters, using the previous chapter only as inspiration. We introduce two approaches, semimeasures and measures, and prove them equivalent. For a more general coverage of probability theory see [Jay03] and [Ash72].

2.1 Sequence notation

We first establish notation for sequences.

Let X be a finite set. A finite sequence[1](or string) over X is a finite list of elements from X, written $x = x_1 x_2 \dots x_n$ for $x_i \in X$ and $n \in \mathbb{N}$. The length of a finite sequence is denoted by $|x|$, equal to n if $x = x_1 \dots x_n$. 00100101 is a finite sequence of length 8 over the set of binary digits $\mathbb{B} = \{0, 1\}$. An infinite sequence over X is an infinite list of elements from X, written $x = x_1 x_2 \dots$ for $x_i \in X$ and $i \in \mathbb{N}$.

$0000\ldots$ is the infinite sequence of all zeros over $\mathbb{B}$. The length of an infinite sequence $x \in X^\infty$ is $|x| = \infty$. The set of finite sequences over X is denoted by X^*, the set of infinite sequences by X^∞, and the set of all sequences by $X^\# = X^* \cup X^\infty$.

We denote the empty sequence by ϵ. This is the only sequence of length 0. The concatenation of two sequences $x, y \in X^\#$, denoted xy, consists of the elements of x followed by the elements of y. If x is infinite then $xy = x$. A sequence $x \in X^\#$ is a prefix of another $y \in X^\#$, denoted $x \preceq y$, if y starts with x. This holds exactly when there exists a $z \in X^\#$ such that $xz = y$. A sequence x is a proper prefix of another y, denoted $x \prec y$, if $x \preceq y$ but not $y \preceq x$. This means y starts with x and includes additional elements. If $x \preceq y$ we can also say y extends x. Finally, given a sequence x the subsequence $x_{i:j} = x_i \ldots x_j$ is the ith through jth elements of x (if $j < i$ we define $x_{i:j} = \epsilon$). For instance, $x_{1:|x|} = x$.

Let $S \subseteq X^\#$ be a set of sequences. The shortest sequence extending every element of S, when it exists, is denoted by $\sup S$. A set is prefix-free if for every $s, t \in S$ such that $s \neq t$ we have neither $t \preceq s$ nor $s \preceq t$. Equivalently, it is prefix-free if for every infinite sequence $\omega \in X^\infty$ there exists at most one $s \in S$ such that $s \preceq \omega$. (See also prefix-free and complete sets in Section 2.3.6.)

2.2 Defining semimeasures

We introduce semimeasures by example, showing how the probability of the propositions "the sequence begins with x" for $x \in X^*$ and "the sequence is exactly z" for $z \in X^\#$ can be calculated from them.

Example 2.1. Suppose we generate an infinite[2] sequence over $\mathbb{B} = \{0, 1\}$ by repeatedly flipping a coin, recording 0 for heads and 1 for tails. The coin is fair so the probability that the next bit is 0 is $1/2$ and the probability that it is 1 is $1/2$. The probability our sequence starts with any particular finite sequence of n zeros and ones is therefore 2^{-n}, no matter which of the 2^n sequences of length n we

[1]We use the same notation for finite and infinite sequence as it is artificial to distinguish between them in our work. This is primarily because processes can output both finite and infinite sequences. We will, for example, not find it necessary to define random sequences where the distinction between finite and infinite is crucial [Cal02].

[2]Infinite as in unbounded: at any point the sequence generated so far is finite, but you can always flip another coin if you need another bit.

choose. We can form a function $\nu_1 \colon X^* \to [0,1]$ taking finite sequences $x \in \mathbb{B}^*$ to the probability the true sequence begins with x:

$$\nu_1(x) = 2^{-|x|}.$$

We will call this particular function the **uniform semimeasure**[3] over $\mathbb{B}$. This corresponds to the prediction that there will always be another coin flip which is equally likely to be heads or tails. It embodies the knowledge that the sequence is unbounded in length, but nothing more. □

Example 2.2. Suppose we generate a sequence over $\mathbb{B} = \{0,1\}$ by repeatedly flipping two coins, recording 0 if the first coin is zero, recording 1 if both coins are one, otherwise stopping. This sequence may be either finite or infinite in extent. At any given point the probability is 1/2 that the next digit is 0, 1/4 that it is 1, and 1/4 that there is no next digit. The probability that the sequence begins with a prefix $x \in X^*$ is therefore

$$\nu_2(x) = 2^{-|x|_0} 4^{-|x|_1}$$

where $|x|_0$ is the number of 0's in x, $|x|_1$ the number of 1's.

The probability that the sequence is *exactly* $x \in X^*$ is

$$\nu_2(x{\downarrow}) = 2^{-|x|_0} 4^{-|x|_1} 4^{-1}.$$

In Example 2.1 we have $\nu_1(x{\downarrow}) = 0$ for all $x \in X^*$ as the generation process creates only infinite sequences. Since every sequence that begins with x is either exactly x, or begins with xb for a unique element $b \in \mathbb{B}$ (but not both), we should have

$$\nu_2(x) = \nu_2(x{\downarrow}) + \sum_{c \in \mathbb{B}} \nu_2(xb)$$

which one can easily verify to be the case. This allows us to compute $\nu_2(x{\downarrow})$ from ν itself:

$$\nu_2(x{\downarrow}) = \nu_2(x) - \sum_{c \in \mathbb{B}} \nu_2(xb).$$

□

[3]See Section 3.2.3 for more on the uniform semimeasure, denoted λ in later chapters.

Example 2.3. Finally, suppose we generate a sequence over $\mathbb{B}$ by flipping a coin once, repeatedly recording 0 if it comes up heads or 1 if it comes up tails. We have,

$$\nu_3(x) = \begin{cases} 1 & \text{if } x = \epsilon, \\ 1/2 & \text{if } x = 0^n \text{ or } x = 1^n \text{ for some } n, \\ 0 & \text{otherwise.} \end{cases}$$

where 0^n is the sequence of n zeros, similarly for 1^n and ones. The sequence is always infinite, so $\nu_3(x{\downarrow}) = 0$ for finite x. The probability that the sequence is the infinite sequence of zeros is $1/2$. We denote this by $\nu_3(0^\infty{\downarrow}) = 1/2$. In general, for any $z \in X^\#$ we have

$$\nu_3(z{\downarrow}) = \begin{cases} 1/2 & \text{if } z = 0^\infty \text{ or } z = 1^\infty, \\ 0 & \text{otherwise.} \end{cases}$$

Note we can derive $\nu_3(z{\downarrow})$ from ν_3 for infinite sequences $z \in \mathbb{B}^\infty$ too:

$$\nu_3(z{\downarrow}) = \lim_{n\to\infty} \nu_3(z_{1:n}).$$

□

The functions ν_1, ν_2, and ν_3 are examples of semimeasures. Their generalisation motivates the formal definition below.

Definition 2.4. A **semimeasure** ν is a function $\nu\colon X^* \to [0,1]$ satisfying:

1. Normalisation:
$$\nu(\epsilon) = 1,$$
2. Coherence:
$$\sum_{c\in X} \nu(xc) \le \nu(x) \quad \text{for all } x \in X^*.$$

□

We interpret $\nu(x)$ as the probability that the actual sequence begins with x. Because all sequences begin with ϵ we have $\nu(\epsilon) = 1$. As any sequence that begins with xc for $c \in X$ cannot begin with xc' for $c' \neq c$, and as it necessarily begins with x, we have $\sum_{c\in X} \nu(xc) \le \nu(x)$.

The probability that a sequence is exactly $z \in X^{\#}$ is denoted $\nu(z{\downarrow})$. As mentioned above, this can be defined from the semimeasure ν by

$$\nu(z{\downarrow}) = \begin{cases} \nu(z) - \sum_{c \in X} \nu(zc) & \text{if } z \in X^*, \\ \lim_{n \to \infty} \nu(z_{1:n}) & \text{if } z \in X^\infty. \end{cases} \tag{2.1}$$

The coherence requirement of Definition 2.4 is equivalent to $\nu(z{\downarrow})$ being nonnegative for all z. Thus, we can restate the coherence requirement:

$$\nu(x) = \nu(x{\downarrow}) + \sum_{c \in X} \nu(xc) \quad \text{for all } x \in X^*. \tag{2.2}$$

where $\nu(x{\downarrow}) \in [0,1]$ for all x.

From the above we see that if $\sum_{c \in X} \nu(xc) = \nu(x)$ holds then $\nu(x{\downarrow}) = 0$. If this equality holds for all $x \in X^*$ then the probability of the sequence being finite is zero. Inequality is required for finite sequences to have nonzero probability.

2.3 Semimeasures are measures over $X^{\#}$

Another way to capture probabilities is through probability measures. A set of sequences $A \subseteq X^{\#}$ is interpreted as the proposition "the true sequences lies within the set A". For example "the true sequence begins with x" corresponds to the set

$$\{y \in X^{\#} : x \preceq y\} = \{xz : z \in X^{\#}\}.$$

This is the set of all sequences consistent with the proposition. A probability measure is a function taking a set of sequences to the probability that the true sequence lies within this set. The probability assigned to the above set corresponds to $\nu(x)$ in the realm of semimeasures. We show probability measures and semimeasures are equivalent ways of recording probabilities.

For technical reasons, probability measures assign probabilities only to particular "measurable" sets of sequences. All propositions we use here will correspond to measurable sets. We require that the set

$$\{xz : z \in X^{\#}\}$$

be measurable for any finite sequence $x \in X^*$. Combining sets by union corresponds to combining propositions by disjunction: $\bigcup_i A_i$ means "at least one of the propositions denoted by A_i is true". Similarly, set complement corresponds to negation. We therefore require measurable sets be closed under complement and countable union[4] (and, as a consequence, closed under countable intersection, set difference, etc).

We will use the following basic sets of sequences:

1. Corresponding to the proposition "the sequence begins with x" we have
$$\Gamma_x = \{xz : z \in X^\#\}$$
for a finite sequence $x \in X^*$. These are called cylinder sets.

2. Corresponding to the proposition "the sequence is exactly x" we have
$$\{x\}$$
for a finite sequence $x \in X^*$. We call these singleton sets.

2.3.1 Measurable sets and probability measures

The set of all measurable sets, here denoted Σ, is closed under countable union and complement. Such sets are called σ-algebras. For later results we will be interested in sets closed under finite union and complement. Such sets are called algebras.

Definition 2.5. An **algebra** $\mathcal{A} \subseteq \mathcal{P}(X^\#)$ is a set of sets of sequences closed under finite union and complement i.e. if $A, B \in \mathcal{A}$ then
$$A \cup B \in \mathcal{A}$$
and if $A \in \mathcal{A}$ then
$$X^\# \setminus A \in \mathcal{A}.$$

[4]Finite union is justified by the above considerations, but it is harder to see why countable union should be allowed. It suffices to say that countable union is a mathematical convenience standardly taken. For example, with countable union and complement we can construct the set $\{z\}$ for $z \in X^\infty$ from the cylinders. Regardless, the results below continue to hold if we restrict ourselves to finite union. See [Ash72] for further discussion.

A **σ-algebra** $\Sigma \subseteq \mathcal{P}(X^\#)$ is an algebra closed under countable union, i.e. if $A_i \in \Sigma$ for $i \in \mathbb{N}$ then

$$\bigcup_i A_i \in \Sigma.$$

□

Note that algebras are closed under set difference and finite intersections, σ-algebras being additionally closed under countable intersections.

We will fix a σ-algebra, denoted Σ, for all of our measures: the smallest σ-algebra containing the cylinder sets

$$\{\Gamma_x : x \in X^*\} = \{\{xz : z \in X^\#\} : x \in X^*\}.$$

Note that the set $\{x\}$ for $x \in X^*$ is measurable as

$$\{x\} = \Gamma_x \setminus \bigcup_{c \in X} \Gamma_{xc}.$$

Probability measures assign probabilities to measurable sets.

Definition 2.6. Given a σ-algebra Σ, a **probability measure** is a function $\phi\colon \Sigma \to [0, \infty)$ such that

$$\phi(X^\#) = 1$$

and

$$\phi\left(\bigcup_i A_i\right) = \sum_i \phi(A_i)$$

for any countable sequence of pairwise disjoint measurable sets $A_i \in \Sigma$. □

As a useful consequence of this, if $A \subseteq B$ then

$$\phi(A) \leq \phi(B)$$

since $B = A \cup (B \setminus A)$. If a proposition A is implied by another B it cannot have greater probability. This implies

$$0 \leq \phi(A) \leq 1$$

for all measurable sets A.

2.3.2 From probability measures to semimeasures and back

Each probability measure corresponds to a unique semimeasure. Given a probability measure ϕ over $X^{\#}$, define the function $\nu\colon X^* \to [0,1]$ by

$$\nu(x) = \phi(\Gamma_x).$$

This is a semimeasure. Firstly, $\nu(\epsilon) = \phi(X^{\#}) = 1$. Secondly, $\sum_c \nu(xc) \leq \nu(x)$ because

$$\bigcup_{c \in X} \Gamma_{xc} \subseteq \Gamma_x$$

and the sets Γ_{xc} are pairwise disjoint.

The remainder of this chapter proves the converse: each semimeasure corresponds to a unique probability measure. Denote by Σ_0 the smallest *algebra* containing the cylinder sets Γ_x. This is a proper subset of the σ-algebra Σ upon which probability measures are defined. We first give an explicit form for the sets within Σ_0. We then show there is a unique countably additive function $\phi\colon \Sigma_0 \to [0,\infty)$ such that

$$\phi(\Gamma_x) = \nu(x) \quad \text{for all } x \in X^*,$$

where a countably additive function satisfies

$$\phi\left(\bigcup_i A_i\right) = \sum_i \phi(A_i)$$

whenever $\bigcup_i A_i \in \Sigma_0$ for a countable sequence of pairwise disjoint sets $A_i \in \Sigma_0$. By Theorems 3.1.4 and 3.1.10 of [Dud89], ϕ uniquely extends to a probability measure over Σ. This establishes our equivalence.

2.3.3 Explicit form for Σ_0 sets

Lemma 2.7. The algebra Σ_0 contains exactly the sets of the form

$$\bigcup_{x \in P} \Gamma_x \cup Q$$

where $P, Q \subseteq X^*$ are finite.

Proof. First note any set of the above form is in Σ_0, and that Γ_x is of the above form. We need only show sets of the above form are closed under finite union and complement. Finite union is trivial, so only complement remains.

Since

$$\begin{aligned}
X^{\#} \setminus \Gamma_x &= \bigcup_{\substack{y \neq x \\ |y|=|x|}} \Gamma_y \ \cup\ \{y \in X^* : |y| < |x|\} \\
X^{\#} \setminus \{x\} &= \bigcup_{|y|=|x|+1} \Gamma_y \ \cup\ \{y \in X^* : |y| \leq |x| \text{ and } y \neq x\}
\end{aligned}$$

and

$$X^{\#} \setminus \left(\bigcup_{x \in P} \Gamma_x \cup Q \right) = \bigcap_{x \in P} \left(X^{\#} \setminus \Gamma_x \right) \cap \bigcap_{x \in Q} \left(X^{\#} \setminus \{x\} \right)$$

to show closure under complement it suffices to show closure under finite intersection. Since $X^{\#}$ is of the correct form we need only show binary intersections:

$$\begin{aligned}
\left(\bigcup_{x \in P} \Gamma_x \cup Q \right) \cap \left(\bigcup_{y \in R} \Gamma_y \cup S \right) &= \left(\bigcup_{x \in P} \Gamma_x \cap \bigcup_{y \in R} \Gamma_y \right) \cup T \\
&= \bigcup_{x \in U} \Gamma_x \cup T
\end{aligned}$$

where $T \subseteq X^*$ is a finite set and

$$\begin{aligned}
U &= \{x : x \in P \text{ and exists } y \in R \text{ where } y \preceq x\} \\
&\cup\ \{y : y \in R \text{ and exists } x \in P \text{ where } x \preceq y\}.
\end{aligned}$$

is likewise a finite set. □

2.3.4 Canonical presentation for Σ_0

Suppose $A \in \Sigma_0$. We define A's canonical form: a minimal decomposition of A into cylinder sets.

For a set $S \subseteq X^*$ denote by $\min S$ its minimal nodes, i.e. all $s \in S$ where there is no $t \in S$ with $t \prec s$.

Definition 2.8. A set $A \in \Sigma_0$ has the **canonical form**

$$A = \bigcup_{x \in C(A)} \Gamma_x \cup S(A)$$

where

$$C(A) = \min\{x \in X^* : \Gamma_x \subseteq A\}$$

and $S(A)$ is defined to make the equality hold. Note that all the sets Γ_x for $x \in C(A)$ and $S(A)$ in the above union are pairwise disjoint. $\square$

Another way to characterise $C(A)$ is that whenever $\Gamma_x \subseteq A$ for $x \in X^*$ then there exists a unique $\hat{x} \in C(A)$ with $\hat{x} \preceq x$ and $\Gamma_x \subseteq \Gamma_{\hat{x}}$. $C(A)$ labels the maximal cylinder sets contained within A.

By Lemma 2.7 we know for any $A \in \Sigma_0$ there exists finite sets $P, Q \subseteq X^*$ such that

$$\bigcup_{x \in C(A)} \Gamma_x \cup S(A) = A = \bigcup_{x \in P} \Gamma_x \cup Q.$$

By definition of $C(A)$, and as X^* is well-founded, for every $x \in P$ there exists a unique $\hat{x} \in C(A)$ such that $\hat{x} \preceq x$. That is, for any Γ_x on the right there is a unique $\Gamma_{\hat{x}}$ on the left such that $\Gamma_x \subseteq \Gamma_{\hat{x}}$. Thus,

$$\bigcup_{x \in P} \Gamma_x \subseteq \bigcup_{x \in C(A)} \Gamma_x,$$

and so $S(A)$ is a finite subset of X^*.

2.3.5 From semimeasures to countable additive functions

We define a countably additive function $\phi\colon \Sigma_0 \to [0, \infty)$ from a semimeasure ν using canonical forms. Given a semimeasure ν define

$$\phi(A) = \sum_{y \in C(A)} \nu(y) + \sum_{y \in S(A)} \nu(y{\downarrow})$$

for each set $A \in \Sigma_0$, recalling that

$$\nu(y{\downarrow}) = \nu(y) - \sum_{c \in X} \nu(yc)$$

which is nonnegative as ν is a semimeasure. Clearly

$$\phi(\Gamma_x) = \nu(x)$$

for all $x \in X^*$.

The remaining sections show ϕ is countably additive i.e. that

$$\phi\left(\bigcup_i A_i\right) = \sum_i \phi(A_i)$$

whenever $\bigcup_i A_i \in \Sigma_0$ for a countable sequence of pairwise disjoint sets $A_i \in \Sigma_0$.

2.3.6 Prefix-free and complete subsets of X^*

Prefix-free and complete sets are central to Lemmata 2.10 and 2.11 below, so we outline a number of results we will require.

A prefix-free and complete (pfc) subset of X^* is a set $P \subseteq X^*$ where every sequence $x \in X^{\#}$ has exactly one $p \in P$ such that either $p \preceq x$ or $x \preceq p$. Equivalently, it is a set where all infinite sequences $\omega \in X^\infty$ have exactly one $p \in P$ prefixing it $p \preceq \omega$. For example $\{00, 01, 1\}$ is a pfc set.

Prefix-free and complete sets $P \subseteq X^*$ are finite. If one weren't then there would be elements of unbounded length in it. Define then

$$F_i = \{\omega \in X^\infty : \text{there is no } p \in P \text{ with } |p| \leq i \text{ such that } p \preceq \omega\}.$$

the set of infinite strings not prefixed by an element of P with length at most i. It is clear that $F_{i+1} \subseteq F_i$, and because P contains elements of unbounded length we know $F_i \neq \emptyset$ for all $i \in N$. But then

$$\bigcap_i F_i$$

is nonempty, so P cannot be complete.

The primary result we use is the following:

Theorem 2.9. Prefix-free and complete (pfc) sets are inductively defined by the following rules.

1. $\{\epsilon\}$ is a pfc set.

2. If P is a pfc set and $x \in P$ then

$$P \setminus \{x\} \cup xX$$

   is a pfc set, where $xX = \{xc : c \in X\}$.

Proof. It is clear that each rule generates a pfc set, for in rule 2 we see $x \preceq \omega$ holds if and only if $xc \preceq \omega$ holds for exactly one $xc \in xX$.

It remains to show the converse. In the following we will partially order pfc sets, writing $P \preceq Q$ if for all $p \in P$ there exists some $q \in Q$ such that $p \preceq q$. We also require an enumeration of finite sequences. Let $f\colon \mathbb{N} \to X^*$ be an onto function such that if $f(x) \preceq f(y)$ then $x \leq y$. This function enumerates finite sequences in an increasing fashion. As an example, f could enumerate $\mathbb{B}^*$ in the order $\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$.

Let P be an arbitrary pfc set and define the sequence P_i of pfc sets by $P_0 = \{\epsilon\}$ and

$$P_{i+1} = \begin{cases} P_i \setminus \{f(i)\} \cup f(i)X & \text{if } f(i) \in P_i \text{ and there exists } p \in P \text{ with } f(i) \prec p, \\ P_i & \text{otherwise.} \end{cases}$$

It is clear that each set P_i is pfc since we start via rule 1 and either don't change the set or apply rule 2. The following observations imply $P = P_n$ for some n:

1. The above sequence is monotonic: for all $p_i \in P_i$ there is a unique $p_{i+1} \in P_{i+1}$ where $p_i \preceq p_{i+1}$.

2. P_{i+1} is bounded above by P. To see this note that if $x \prec p$ for $p \in P$ then for all $c \in X$ we have $xc \preceq p'$ for some $p' \in P$.

3. Every prefix of an element of P appears in some P_i. Trivial for ϵ. Otherwise let $x \preceq p$ for some $p \in P$, and $x = x'c$ for some $c \in X$. We inductively assume x' first appears in P_{i+1}, so $f(i) = x'$. Then some $j > i$ has $f(j) = x'c = x$, and so P_{j+1} contains x.

4. The sequence P_i is eventually constant, since eventually $f(i)$ is longer than all of P.

□

2.3.7 Establishing countable additivity

The following two lemmata will be required.

Lemma 2.10. If

$$\Gamma_x = \bigcup_{y \in P} \Gamma_y \cup Q$$

for $x \in X^*$ and $P, Q \subseteq X^*$, where the sets Γ_y for $y \in P$ and Q on the right are pairwise disjoint, then there exists a pfc set $E \subseteq X^*$ such that

$$P = xE = \{xe : e \in E\}$$

and

$$Q = x^E = \{xd : d \in X^*, \text{ and } d \prec e \text{ for some } e \in E\}.$$

Proof. Let

$$E = \{e \in X^* : xe \in P\}$$

be the labels of all the cylinders from P minus their common prefix x. This set is prefix-free and complete. To see this note that for any $\omega \in X^\infty$ we have $x\omega \in \Gamma_x$. As $\bigcup_{y \in P} \Gamma_y$ is a disjoint cover of the infinite sequences in Γ_x, there is exactly one Γ_y containing $x\omega$ and so exactly one $y \in P$ with $y \preceq x\omega$. As a result there is exactly one $e \in E$ such that $e \preceq \omega$, namely the e such that $xe = y$.

It remains to show that

$$\{xd : d \in X^*, \text{ and } d \prec e \text{ for some } e \in E\}$$

equals Q. All elements xd of Q must be in the above set, for if $e \preceq d$ for some $e \in E$ then $xd \in \Gamma_{xe}$ contradicting disjointness, and by completeness of E if $e \preceq d$ doesn't hold for any e then $d \prec e$ holds for some e. Conversely all elements in this set must be elements of Q as although $xd \in \Gamma_x$ it is not contained in any cylinder set for otherwise $e' \preceq d$ for some $e' \in E$ which is impossible as E is prefix-free. □

Lemma 2.11. For any pfc set $E \subseteq X^*$ we have

$$\nu(x) = \sum_{y \in xE} \nu(y) + \sum_{y \in x^E} \nu(y{\downarrow})$$

where

$$x^E = \{xd : d \in X^*, \text{ and } d \prec e \text{ for some } e \in E\}.$$

Proof. We show this by induction over the pfc set E (recall Theorem 2.9).

This is trivial for $\{\epsilon\}$ as $x\{\epsilon\} = \{x\}$ and $x^{\{\epsilon\}} = \emptyset$.

Suppose $E = E' \setminus \{z\} \cup zX$ where $z \in E'$. As

$$\begin{aligned} xE &= xE' \setminus \{xz\} \cup (xz)X \\ x^E &= x^{E'} \cup \{xz\} \end{aligned}$$

and as

$$\nu(xz) = \sum_{y \,\in\, (xz)X} \nu(y) + \nu(xz{\downarrow})$$

by Equation (2.2), we have

$$\begin{aligned} & \sum_{y \in xE} \nu(y) + \sum_{y \in x^E} \nu(y{\downarrow}) \\ = & \sum_{y \in xE'} \nu(y) - \nu(xz) + \sum_{y \,\in\, (xz)X} \nu(y) + \nu(xz{\downarrow}) + \sum_{y \in x^{E'}} \nu(y{\downarrow}) \\ = & \sum_{y \in xE'} \nu(y) + \sum_{y \in x^{E'}} \nu(y{\downarrow}). \end{aligned}$$

□

Finally, we can prove our desired equivalence between semimeasures and probability measures. In the following theorem certain identities only hold beacuse unions are disjoint and set differences are proper. We sometimes skip the details for clarity, although they can be easily verified from the surrounding context.

Theorem 2.12. For every semimeasure $\nu\colon X^* \to [0,1]$ there is a unique probability measure $\phi\colon \Sigma \to [0,1]$ such that

$$\phi(\Gamma_x) = \nu(x) \quad \text{for all } x \in X^*.$$

Conversely, for every probability measure there is a unique semimeasure such that the above holds.

Proof. The measure to semimeasure direction was established in Section 2.3.2.

Given a semimeasure ν we define the function $\phi\colon \Sigma_0 \to [0,\infty)$ by

$$\phi(A) = \sum_{y \in C(A)} \nu(y) + \sum_{y \in S(A)} \nu(y{\downarrow})$$

for each set $A \in \Sigma_0$, following Section 2.3.5. By the discussion in Section 2.3.2 we need only show countable additivity to establish that ϕ uniquely extends to the desired measure.

Suppose, then, that

$$A = \bigcup_i A_i$$

where $A_i \in \Sigma_0$ for $i \in \mathbb{N}$ are pairwise disjoint, and $A \in \Sigma_0$. By Definition 2.8 of canonical forms we have

$$\bigcup_{y \in C(A)} \Gamma_y \cup S(A) = \bigcup_i \bigcup_{y_i \in C(A_i)} \Gamma_{y_i} \cup S(A_i)$$

where the sets on either side are pairwise disjoint. By the discussion in Section 2.3.4 any Γ_{y_i} on the right is a subset of exactly one Γ_y on the left. Similarly, each element in $S(A_i)$ lies either in a unique Γ_{y_i} on the right or within $S(A)$. Therefore, for all $y \in C(A)$

$$\Gamma_y = \bigcup_{y' \in B_y} \Gamma_{y'} \cup R_y$$

where

$$B_y = \{y' : y' \in C(A_i) \text{ for some } i \text{ and } \Gamma_{y'} \subseteq \Gamma_y\}$$

lists of all the cylinders Γ_{y_i} within Γ_y and

$$R_y = \{x : x \in S(A_i) \text{ for some } i \text{ and } x \in \Gamma_y\}$$

contains all the elements of $S(A_i)$ within Γ_y.

By lemmata 2.10 and 2.11 this means

$$\nu(y) = \sum_{y' \in B_y} \nu(y') + \sum_{x \in R_y} \nu(x\downarrow).$$

But $\bigcup_{y \in C(A)} B_y = \bigcup_i C(A_i)$ so

$$\begin{aligned} \sum_{y \in C(A)} \nu(y) &= \sum_{y \in C(A)} \left(\sum_{y' \in B_y} \nu(y') + \sum_{x \in R_y} \nu(x\downarrow) \right) \\ &= \sum_i \sum_{y_i \in C(A_i)} \nu(y_i) + \sum_{y \in C(A)} \sum_{x \in R_y} \nu(x\downarrow). \end{aligned}$$

Since

$$S(A) = \left(\bigcup_i S(A_i) \right) \setminus \left(\bigcup_{y \in C(A)} R_y \right)$$

we have

$$\sum_{y \in C(A)} \nu(y) + \sum_{y \in S(A)} \nu(y\downarrow) = \sum_i \sum_{y_i \in C(A_i)} \nu(y_i) + \sum_i \sum_{y_i \in S(A_i)} \nu(y_i\downarrow)$$

and so

$$\phi(A) = \sum_i \phi(A_i).$$

$\square$

Chapter 3

Computability and prediction

Computability [Odi89] is a necessary property of those systems we can build[1] [Gan80] and perhaps even reality itself [Fre90]. This motivates interest both in predicting computable systems and computing predictions. These two approaches are complementary: one applies computability to that which is predicted, the other to that which predicts. This chapter describes these two approaches and proves them equivalent. This equivalence will allow us to construct a universal semimeasure in Chapter 4.

The key definitions of this chapter are processes (Definition 3.6 in Section 3.2) and enumerable semimeasures (Definition 3.13 in Section 3.3). The key result is their equivalence (Theorem 3.29 in Section 3.4).

Processes are computable systems (Section 3.1 briefly describes the computability theory we use). As a process's output is completely determined by its input, we can derive predictions about a process's output from knowledge about its input. In other words, from a semimeasure over a process's input we uniquely derive a semimeasure over its output. There is reason to focus on the output semimeasure derived by using the uniform semimeasure $\lambda(p) = 2^{-|p|}$ over the input. This semimeasure, which we will call the process's semimeasure, is the knowledge we have of the process's output when it is fed unbounded but unknown bits (Section 3.2.3).

A semimeasure is enumerable when it can be computed from below. Enumerable semimeasures are interesting because they are exactly the semimeasures of processes (Section 3.4). That is, any enumerable semimeasure can be seen as the knowledge

[1]We have yet to build anything uncomputable.

we have about the output of some process that is fed completely unknown but unbounded input (Section 3.2.3). Alternatively, we can think of the process being fed a completely random input stream, the semimeasure giving the probability for each possible output. We can effectively convert between these two forms: process and enumerable semimeasure.

One interpretation of this equivalence (see Section 3.4.6) is that partial knowledge about a sequence (the original semimeasure) can be extracted to form complete knowledge of a process leaving behind complete ignorance of another sequence (the uniform semimeasure).

3.1 Computability of sets

Our introduction to computability will be brief, see [Odi89] for further details. The key definition is that of a computably enumerable set.

The most basic notion of computability is that of **computable functions** between countable sets. A function $f\colon A \to B$ is computable if there is an algorithm which given an $a \in A$ returns $f(a) \in B$ in finite time. Informally one can think of an algorithm as a sequence of instructions that can be executed in finite time, such as a computer program. Algorithms can be formally defined in many essentially equivalent ways: Turing machines, register machines, lambda calculus terms, etc. We will describe algorithms using informal pseudocode, leaving implicit their translation into formal models.

A partial function $f\colon A \overset{o}{\to} B$ is one where $f(a)$ need not be defined for all $a \in A$, although it has a unique value when it is. Computability is typically extended to cover **computable partial functions** (or partial computable functions). These are implemented by algorithms that which given a return $f(a)$ if it is defined otherwise returning nothing. This extension is important because, in general, algorithms may fail to halt.

Computability can be abstracted from functions to sets. A subset $S \subseteq A$ of some fixed countable set A is **computable** if there is a computable function $\chi_S\colon A \to \mathbb{B}$ such that $\chi_S(a) = 1$ if and only if $a \in S$. In other notation, if $\chi_S^{-1}(1) = S$. This means we have an algorithm which can test whether $a \in S$ or not in finite time.

A subset $S \subseteq A$ is **computably enumerable** if there is a computable *partial* function $f\colon A \overset{o}{\to} \mathbb{B}$ such that $f(a) = 1$ if and only if $a \in S$. This means we have an algorithm that on input of $a \in A$

1. Returns 1 if $a \in S$,
2. Returns 0 or nothing (i.e. $f(a)$ is undefined) if $a \notin S$.

With this we can test whether $a \in S$, but if $a \notin S$ the algorithm may fail to terminate. The option of failing to terminate is useful if S is a set of the form

$$S = \{a \in A : \text{there exists } c \in C \text{ such that } \langle a, c\rangle \in T\}$$

for some computable set T. That is, if the algorithm makes a potentially infinite search which always succeeds if $a \in A$.

Equivalently, a subset $S \subseteq A$ is computably enumerable if there exists an enumeration of S i.e. an function $e\colon \mathbb{N} \to A$ such that $e(\mathbb{N}) = S$. This function lists all elements of S, possibly with repetition. Given an enumeration of S we have

$$S = \{a \in A : \text{there exists } n \in \mathbb{N} \text{ such that } e(n) = a\}$$

so S is computably enumerable. Conversely, we can form an enumeration of S from a partial function $f\colon A \overset{o}{\to} \mathbb{B}$ such that $f^{-1}(1) = S$ by, roughly speaking, evaluating f on all members of $a \in A$ in parallel outputting elements $a \in A$ whenever we compute $f(a) = 1$.

We will need to define computable functions between uncountable sets, such as monotone functions $F\colon \mathbb{B}^{\#} \to X^{\#}$ and semimeasures $\nu\colon X^* \to [0,1]$. Computability of these uncountable objects can be reduced to the computability of certain countable subsets derived from the object. See Definitions 3.6 and 3.13 for examples of this.

3.2 Processes: computable systems

A system is anything with input and output streams. A vending machine is a system which inputs money and directions and outputs things. A computer is a system which inputs data and outputs results. With Y denoting the possible inputs a system could receive at any given instant, and X the set of possible outputs, a system's behaviour is a function $M\colon Y^{\#} \to X^{\#}$ recording the sequence $F(y)$ the system outputs whenever[2] it is input the sequence $y \in Y^{\#}$.

A computable system is a system with computable behaviour. A process [Cal02] is the behaviour of a computable system. We assume processes input bits[3] i.e. that $Y = \mathbb{B} = \{0,1\}$. Processes can be thought of as computers which input data and output results.

Section 3.2.1 formally defines processes. Section 3.2.2 motivates the formal definition by showing processes are equivalent to interactive algorithms. Section 3.2.3 defines a process's semimeasure.

3.2.1 Defining processes

We introduce processes by way of example.

Example 3.1. Consider a system which inputs bits from $\mathbb{B} = \{0,1\}$ and outputs symbols from $X = \{+,-\}$. Every pair of bits read encodes a particular action: output a character, do nothing, or halt (see Table 3.1). The box keeps on reading input bits until it reads the halt code. If it tries to read a bit and there is none waiting it pauses until there is.

Code	Action
00	Output $+$
11	Output $-$
01	Halt execution
10	Do nothing

Table 3.1: Codes for process M_1

Define a function $M_1\colon \mathbb{B}^{\#} \to X^{\#}$ capturing the behaviour of this system: $M_1(p)$ is the output of the system when given $p \in \mathbb{B}^{\#}$ as input (see Table 3.2 for sample values).

This function has two interesting properties:

[2]We assume the output is determined completely by the input.

[3]Although all results generalise to arbitrary finite Y, for our purposes such generalisation adds unnecessary complexity.

p	$M(p)$
0	ϵ
00	$+$
0011	$+-$
0010110111	$+-$
$000000\ldots$	$+++\cdots$
$010000\ldots$	ϵ

Table 3.2: Sample values of M_1

1. Monotonicity. If $p \preceq q$ then $M_1(p) \preceq M_1(q)$. This corresponds to temporal consistency: adding further inputs cannot change previous outputs.

2. Continuity. If $\omega \in \mathbb{B}^\infty$ then

$$M_1(\omega) = \sup\{M_1(p) : p \prec \omega\}$$

where $\sup S$ denotes the shortest sequence extending every element of $S \subseteq X^\#$. This means every symbol in the output is output at some finite time.

$\square$

In general the behaviour of all systems satisfy monotonicity and continuity.

Definition 3.2. A **monotone function** M is a function $M\colon \mathbb{B}^\# \to X^\#$ such that for any $p_1, p_2 \in \mathbb{B}^\#$ if $p_1 \preceq p_2$ then

$$M(p_1) \preceq M(p_2).$$

$\square$

Definition 3.3. A monotone function $M\colon \mathbb{B}^\# \to X^\#$ is **continuous** if for all $\omega \in \mathbb{B}^\infty$ we have

$$M(\omega) = \sup\{M(p) : p \prec \omega\}.$$

$\square$

Example 3.4. Consider another system. On input of i zeros followed by a one (i.e. 0^i1) the system simulates the ith program[4]. If the program halts, the system outputs $+$ then reads in another program. Otherwise the system loops forever outputting nothing more. For instance, suppose the 2nd program halts but the 3rd does not. Then the system outputs + when input either 0010001 or 0010001001.

The function $M_2 \colon \mathbb{B}^{\#} \to X^{\#}$, where $M_2(p)$ is the output of this system given input p, is monotone and continuous as in Example 3.1 above. It satisfies two further properties:

1. If we want to know whether $M_2(p)$ begins with a string $x \in X^*$ we can write an algorithm which will output 1 if and only if it does: simulate the above process and output 1 if the simulated output begins with x. That is, the set

   $$G_{M_2} = \{\langle p, x\rangle \in \mathbb{B}^* \times X^* : x \preceq M_2(p)\}$$

   is computably enumerable (see Section 3.1). This set is not computable since $+ \preceq M_2(0^i1)$ holds if and only if the ith program halts.

2. The set

   $$\{\langle p, x\rangle \in \mathbb{B}^* \times X^* : x = M_2(p)\}$$

   is not even computably enumerable. We know that either $M_2(0^i1) = +$ or $M_2(0^i1) = \epsilon$ holds, the first if the ith program halts, the second if it doesn't. Thus, were the above set computably enumerable we could solve the halting problem.

   However, the set

   $$H_{M_2} = \{\langle p, x\rangle \in \mathbb{B}^* \times X^* : x = M_2(p) \text{ and } \exists p' \succ p \text{ where } M_2(p') \succ M_2(p)\}$$

   is computably enumerable. Suppose we are trying to decide if $x = M_2(p)$. If there is a $p' \succ p$ such that $M_2(p') \succ M_2(p)$ then all the programs encoded in p must halt. Therefore any algorithm to decide whether $x = M_2(p)$ can wait until all the programs in p halt before checking whether the simulated process outputs x. This avoids the problem with the previous set.

Note that M_1 in Example 3.1 also satisfies these two properties, although in this case G_{M_1} and H_{M_1} are also computable. See the proof of Theorem 3.7 for another example illustrating the above sets H_{M_2} and G_{M_2}. $\square$

[4]See [Odi89] for proof that we can label programs with numbers.

In general all computable systems satisfy the above properties. This leads to the definition of a process.

Definition 3.5. A string $p \in \mathbb{B}^*$ is a **non-terminal string** of the monotone function M if there exists a string $p' \succ p$ such that

$$M(p') \succ M(p).$$

□

Note that if p is a non-terminal string then $M(p)$ is a finite string.

Definition 3.6. A **process** M is a continuous monotone function $M\colon \mathbb{B}^\# \to X^\#$ where the following sets are computably enumerable:

1. The set of finite strings which prefix the output of finite programs

$$G_M = \{\langle p, x\rangle \in \mathbb{B}^* \times X^* : x \preceq M(p)\}.$$

2. The output of non-terminal strings

$$H_M = \{\langle p, M(p)\rangle \in \mathbb{B}^* \times X^* : \ p \text{ is a non-terminal string of } M\}.$$

□

3.2.2 Processes are equivalent to interactive algorithms

This section offers further motivation for the definition of a process by showing processes are exactly the behaviours of interactive algorithms.

Let an **interactive algorithm**[5] be an imperative model of computation expanded with two instructions:

1. $b \leftarrow \text{Input}()$. This inputs a bit $b \in \mathbb{B}$.
2. $\text{Output}(c)$. This outputs a symbol $c \in X$.

For example the pseudocode in Algorithm 1 is an interactive algorithm which outputs a '+' whenever a zero is read. By an analog of the Church-Turing thesis [Odi89] any model of computation will do. For instance, monotone Turing machines [Hut05] [Sch02] implement exactly interactive algorithms. We show interactive algorithms and processes are equivalent: the behaviour of an interactive algorithm is a process and any process can be implemented by an interactive algorithm. Following Section 3.1 we are content to describe algorithms with informal pseudocode.

Algorithm 1 Example interactive algorithm

```
1: loop
2:     $b \leftarrow$ Input().
3:     if $b = 0$ then
4:         Output(+).
5:     end if
6: end loop
```

Theorem 3.7. For a function $M\colon \mathbb{B}^\# \to X^\#$ the following are equivalent:

1. M can be implemented by an interactive algorithm.
2. M is a process.

Proof. Recall Definition 3.6: a process is a continuous monotone function $M\colon \mathbb{B}^\# \to X^\#$ such that the sets

1. $G_M = \{\langle p, x\rangle \in \mathbb{B}^* \times X^* : x \preceq M(p)\}$
2. $H_M = \{\langle p, x\rangle \in \mathbb{B}^* \times X^* : x = M(p) \text{ and } \exists p' \succ p \text{ where } M(p') \succ M(p)\}$

are computably enumerable.

Suppose M can be implemented by an interactive algorithm A. We can determine whether $x \preceq M(p)$ by running A with p as its input, outputting 1 if A eventually outputs x. So G_M is computably enumerable. We can determine $x = M(p)$ for non-terminal p by running A on input p, waiting for it to request more than $|p|$ input bits. If it does so, we output 1 if and only if the output so far is exactly x. So H_M is computably enumerable. Continuity and monotonicity are trivial.

Conversely, suppose M is a process. Algorithm 2, an interactive algorithm, implements M. To see this let $D\colon \mathbb{B}^\# \to X^\#$ be the behaviour of Algorithm 2.

[5]Not to be confused with the interactive algorithms of [Gur05].

Algorithm 2 Implementation of M

```
1: $p_0 \leftarrow \epsilon$, $x_0 \leftarrow \epsilon$.
2: loop
3:     Enumerate $G_M$ and $H_M$ simultaneously, yielding an element $\langle p, x\rangle$ of one.
4:     if $p_0 = p$ and $x_0 \preceq x$ then
5:         Let $t \in X^*$ be the symbols in $x$ but not $x_0$ i.e. $t$ such that $x_0 t = x$.
6:         Output $t$.
7:         $x_0 \leftarrow x$.
8:         if $\langle p, x\rangle \in H_M$ then
9:             $b \leftarrow$ Input().
10:            Set $p_0 \leftarrow p_0 b$.
11:        end if
12:    end if
13: end loop
```

Fix a sequence of bits $\hat{p} \in \mathbb{B}^*$ to be input to Algorithm 2. In the following $\hat{p}$ will always denote this fixed sequence, p_0 will denote the current value of the variable p_0, and $p \in \mathbb{B}^*$ will be an arbitrary binary string.

First note that both $x_0 \preceq M(p_0)$ and $x_0 \preceq D(p_0)$ hold for all values the variables x_0 and p_0 take. Also, $D(p_0) = M(p_0)$ holds if step 9 is reached.

Fix a string $p \in \mathbb{B}^*$. If at some point in time the variable p_0 equals p, then $D(p) = M(p)$. To see this we will show $x \preceq M(p)$ implies $x \preceq D(p)$ for all x.

Suppose $x \preceq M(p)$. The pair $\langle p, x\rangle$ will eventually be enumerated[6] in step 3 whilst the variable p_0 equals p unless step 9 occurs first and changes p_0. If step 9 occurs first then we have $x \preceq M(p) = D(p)$ as by previous observation $M(p_0) = D(p_0)$ if step 9 is reached. Otherwise, when the pair $\langle p, x\rangle$ is enumerated in step 3 we know the variable p_0 equals p, and that either $x_0 \preceq x$ or not. If $x_0 \preceq x$ then as step 6 ensures that $x \preceq D(p_0)$ we have $x \preceq D(p_0) = D(p)$. If not since $x_0 \preceq M(p_0)$ always holds, and as $x \preceq M(p) = M(p_0)$, we have $x \prec x_0$. Thus since $x_0 \preceq D(p_0)$ always holds, we have $x \prec x_0 \preceq D(p_0) = D(p)$. In all cases $x \preceq D(p)$.

If $\langle p, x\rangle \in H_M$ for some x we say p is *nonterminal.* For all $p \in \mathbb{B}^*$ we inductively show

If $p \preceq \hat{p}$ and all proper prefixes $p' \prec p$ of p are nonterminal,
then at some point in time the variable p_0 equals p.

[6]Without loss of generality, we assume every element of G_M and H_M is enumerated infinitely often in step 3 (this is easy to ensure).

This is trivially true for $p = \epsilon$. Suppose that the statement is true for p, that $pb \preceq \hat{p}$, and that all proper prefixes $p' \prec pb$ of pb are nonterminal (if either of the latter two conditions are false the statement is vacuously true). By inductive hypothesis at some point the variable p_0 equals p, and $\langle p, x \rangle \in H_M$ for some x. Eventually step 3 will enumerate $\langle p, x \rangle$ from H_M, and so b will be input in step 10, and the variable p_0 will be set to pb.

Finally, note there is a greatest $p \preceq \hat{p}$ such that all proper prefixes $p' \prec p$ of p are nonterminal. This is because this holds for ϵ and the elements are bounded above by $\hat{p}$. Call the greatest such string p^*.

By the above discussion we have $D(p^*) = M(p^*)$ since the variable p_0 takes the value p^* at some point. If $p^* = \hat{p}$ then $D(\hat{p}) = M(\hat{p})$ holds trivially. Otherwise the greatest element p^* is not nonterminal, so by definition of H_M we know $M(\hat{p}) = M(p^*)$ since $\hat{p} \succ p^*$ and M is monotone. As p^* is not nonterminal we know step 9 is never executed after the variable p_0 is set to p^*, and so $D(\hat{p}) = D(p^*)$. Thus, $D(\hat{p}) = D(p^*) = M(p^*) = M(\hat{p})$.

Since $\hat{p}$ was arbitrary we know $D = M$, so Algorithm 2 implements M. □

3.2.3 The semimeasure generated by a process

As mentioned in this chapter's introduction, a semimeasure over the input of a process uniquely determines a semimeasure over the output of the process. Definition 3.8 and Lemma 3.9 below define the probability measure equivalent to this semimeasure, making use of the equivalence between semimeasures and probability measures established in Theorem 2.12. After giving justification for the use of the uniform semimeasure λ, Definition 3.10 defines a process's semimeasure. This definition is not suitable for computation, so Lemma 3.11 gives an explicit form.

Conventions

Lemmata 3.9, 3.11, and later results will follow [Knu92] in using the notation

$$\sum_x f(x)\ [P(x)] = \sum_{x\colon P(x)} f(x)$$

and analogously

$$\bigcup_x f(x)\ [P(x)] = \bigcup_{x\colon P(x)} f(x).$$

This notation allows us to manipulate complex predicates inline rather than in subscript.

We will use probability measures and semimeasures interchangeably by the equivalence proved in Section 2.3.2, denoting both by the same symbol where this causes no confusion.

The image measure ϕM^{-1} of a measure ϕ

If the probability measure ϕ over $\mathbb{B}^\#$ describes our knowledge of the input sequence of a process M, then the image measure ϕM^{-1} over $X^\#$ describes our knowledge of the output sequence.

Definition 3.8. Given a process $M\colon \mathbb{B}^\# \to X^\#$ and a probability measure ϕ over $\mathbb{B}^\#$ the **image measure** ϕM^{-1} over $X^\#$ is defined by

$$(\phi M^{-1})(A) = \phi(M^{-1}(A)) = \phi(\{p \in \mathbb{B}^\# : M(p) \in A\})$$

for all measurable sets $A \subseteq \mathbb{B}^\#$. □

This is the only possible definition as M's output lies within A exactly when $M(p) \in A$. The probability of this is $\phi(\{p \in \mathbb{B}^\# : M(p) \in A\})$.

Lemma 3.9. For any process M and probability measure ϕ, the image measure ϕM^{-1} is a probability measure.

Proof. Note that

$$M^{-1}(X^\#) = \mathbb{B}^\#$$

and for a countable sequence $A_i \subseteq X^\#$ of pairwise disjoint measurable sets

$$M^{-1}\left(\bigcup_i A_i\right) = \bigcup_i M^{-1}(A_i)$$

where the sets on the right are pairwise disjoint. Thus, by the Definition 2.6 of probability measures it suffices to show that $M^{-1}(A)$ is a measurable set whenever

A is. Since we also have

$$M^{-1}(X^{\#} \setminus A) = \mathbb{B}^{\#} \setminus M^{-1}(A)$$

by the Definition 2.5 of σ-algebras it suffices to show that $M^{-1}(\Gamma_x)$ is measurable for all $x \in X^*$, where $\Gamma_x = \{xz : z \in X^{\#}\}$.

First note that

$$M^{-1}(\Gamma_x) = \{p \in \mathbb{B}^{\#} : x \preceq M(p)\}.$$

Suppose $x \preceq M(p)$ for an infinite sequence $p \in \mathbb{B}^{\infty}$. As x is finite, by continuity of M there must exist a finite prefix $p_f \prec p$ of p with $x \preceq M(p_f)$. Thus, by monotonicity of M

$$\{p \in \mathbb{B}^{\#} : x \preceq M(p)\} = \bigcup_{p_f \in \mathbb{B}^*} \{p \in \mathbb{B}^{\#} : p_f \preceq p\}[x \preceq M(p_f)].$$

The right hand set is measurable as it is a countable union of measurable sets. □

A process's semimeasure

This thesis focuses on the image measure λM^{-1} of a process M, where λ is the probability measure equivalent to the uniform semimeasure

$$\lambda(p) = 2^{-|p|} \quad \text{for all } p \in \mathbb{B}^*.$$

This is because our main goal is to construct a universal semimeasure in Chapter 4. Given an enumerable semimeasure ν we will want to find a process which has ν as its output semimeasure when fed input according to λ. See Sections 4.2.2 and 4.2.4.

The probability measure λ corresponds to knowing only that the sequence is unbounded, nothing more. It is the knowledge that M is fed an unbounded sequence of completely unknown bits. Or, more controversially, the knowledge that M is fed "randomly generated" bits (e.g. from a quantum mechanical noise source). This will have particular significance in Section 3.4.6. To see this, we use the group invariance method of [Jay68] (see especially p38 of [Jay03]).

Suppose we discover that the second bit of the sequence has been inverted e.g. if the sequence previously began with 001010 it now begins with 011010 and vice versa. Letting λ denote the semimeasure corresponding to the original state of knowledge and λ' denote that corresponding to the new state we have

$$\begin{aligned}\lambda(001010) &= \lambda'(011010),\\ \lambda(011010) &= \lambda'(001010).\end{aligned}$$

Although we know something extra about how the sequence is generated if we know the second bit is flipped, we do not know anything extra about the sequence itself. As the same knowledge corresponds to the same probabilities we have

$$\begin{aligned}\lambda(001010) &= \lambda'(001010),\\ \lambda(011010) &= \lambda'(011010).\end{aligned}$$

Combining both sets of equations implies

$$\lambda(001010) = \lambda(011010).$$

Generalising the above, we find that

$$\lambda(p) = \lambda(q)$$

for all sequences $p, q \in \mathbb{B}^*$ of the same length $|p| = |q|$. So $\lambda(p) = f(|p|)$ for some $f\colon \mathbb{N} \to [0, 1]$. Since we know the sequence is unbounded we have $\lambda(p{\downarrow}) = 0$ for all $p \in \mathbb{B}^*$ and thus require

$$f(|p|) = \lambda(p) = \lambda(p{\downarrow}) + \sum_{b\in\mathbb{B}} \lambda(pb) = 2f(|p| + 1)$$

Noting that $f(0) = 1$, the only solution to this identity is $f(n) = 2^{-n}$ giving our result.

The above motivates the definition of a process's semimeasure $\mu_M(x)$.

Definition 3.10. Given a process $M\colon \mathbb{B}^\# \to X^\#$, its semimeasure μ_M is the unique semimeasure equivalent to the image measure λM^{-1}. That is,

$$\mu_M(x) = \lambda M^{-1}(\Gamma_x)$$

for all $x \in X^*$. □

Practical form for a process's semimeasure

Definition 3.10 is not suitable for computational purposes so Lemma 3.11 below gives an explicit form.

Lemma 3.11. If M is a process then

$$\mu_M(x) = \sum_p 2^{-|p|} \, [p \in \min\{p \in \mathbb{B}^* : x \preceq M(p)\}]$$

for any finite sequence $x \in X^*$.

Proof. By definition

$$\mu_M(x) = \lambda M^{-1}(\Gamma_x).$$

Suppose $p \in M^{-1}(\Gamma_x)$ is an infinite sequence. We have $x \preceq M(p)$. As x is finite, by continuity of M there must exist a finite prefix $p_f \prec p$ of p with $x \preceq M(p_f)$. But then by monotonicity of M we have

$$p \in \Gamma_{p_f} \subseteq M^{-1}(\Gamma_x).$$

So every sequence is contained in a cylinder set, and as a result

$$M^{-1}(\Gamma_x) = \bigcup_{p \in \mathbb{B}^*} \Gamma_p [p \in \min\{p \in \mathbb{B}^* : \Gamma_p \subseteq M^{-1}(\Gamma_x)\}].$$

Furthermore all the above sets are pairwise disjoint since if neither $p \preceq q$ nor $q \preceq p$ then $\Gamma_p \cap \Gamma_q = \emptyset$. We then have

$$\begin{aligned} \lambda M^{-1}(\Gamma_x) &= \sum_p \lambda(\Gamma_p)[p \in \min\{p \in \mathbb{B}^* : \Gamma_p \subseteq M^{-1}(\Gamma_x)\}] \\ &= \sum_p 2^{-|p|}[p \in \min\{p \in \mathbb{B}^* : x \preceq M(p)\}]. \end{aligned}$$

The first step since λ is a measure, the last by definition of λ and since M is monotone. □

3.3 Enumerable semimeasures

There are a number of different ways to make semimeasures computable. One natural approach is to call a semimeasure ν computable if we can compute the binary expansion of $\nu(x)$ to arbitrary precision for any x. By this we mean the function

$$f\colon X^* \times \mathbb{N} \to \mathbb{B}^*$$

is computable in the sense of Section 3.1, where $f(x, n)$ is the first n bits of the value of $\nu(x)$ i.e. if $\nu(x) = 0.b_1 b_2 \dots b_i \dots$ in binary ($b_i \in \mathbb{B}$; this is always possible since $\nu(x) \leq 1$) then $f(x, n) = b_1 \dots b_n$.

Since the rational numbers $\mathbb{Q}$ are countable, we could instead ask for a function

$$g\colon X^* \times \mathbb{Q} \to \mathbb{Q}$$

which computes rationals arbitrarily close to $\nu(x)$ i.e. such that $|g(x, \varepsilon) - \nu(x)| < \varepsilon$ holds for all $x \in X^*$ and $\varepsilon \in \mathbb{Q}$.

Both definitions above are equivalent to the following.

Definition 3.12. A semimeasure ν is **computable** if the sets

$$\{\langle \alpha, x\rangle \in \mathbb{Q} \times X^* : \alpha < \nu(x)\}$$

and

$$\{\langle \alpha, x\rangle \in \mathbb{Q} \times X^* : \nu(x) < \alpha\}$$

are computably enumerable. □

This definition, although natural, is too strong: a process's semimeasure need not be computable. For instance, the semimeasure of M_2 from Example 3.4 above is not computable. To see this note that

$$\mu_{M_2}(+) = \sum_i 2^{-(i+1)}[\text{program } i \text{ halts}].$$

If we could compute $\mu_{M_2}(+)$ to arbitrary precision we could solve the halting problem.

A weaker form of computability is required. A semimeasure ν is enumerable, or semicomputable from below, if there exists a function $h\colon X^* \times \mathbb{N} \to \mathbb{Q}$ such that

$$\nu(x) = \lim_{n\to\infty} h(x, n)$$

and where h is nondecreasing i.e. $h(x, n) \leq h(x, m)$ if $n \leq m$. This means we can approximate ν's value arbitrarily closely from below, although unlike computable semimeasures we can never know how close this approximation is. This is equivalent to the following.

Definition 3.13. A semimeasure ν is **enumerable** if the set

$$\{\langle \alpha, x\rangle \in \mathbb{Q} \times X^* : \alpha < \nu(x)\}$$

is computably enumerable. $\square$

It's important that the inequality is strict in this definition. Construct a process M_p that on input $p0^i1$ runs the program p for i steps outputting ϵ if it halts in that time or p otherwise (we assume the programs are encoded in a prefix-free fashion; see Section 4.2.4). As a result $2^{-|p|} \leq \mu_M(p)$ holds iff p never halts. If we had an algorithm enumerating

$$\{\langle \alpha, x\rangle \in \mathbb{Q} \times X^* : \alpha \leq \mu_M(x)\}$$

would we be able to solve the halting problem, an impossibility. So for some processes the above set is not computably enumerable. As the rest of this chapter will demonstrate, with strict inequality this problem dissolves.

3.4 Processes and semimeasures are equivalent

Enumerable semimeasures are exactly those generated as the output semimeasure of a process. Although it is easy to show the semimeasure of a process is enumerable, the converse is not as simple.

To prove the converse we first establish a method for building processes. A process satisfies a request $r = \langle p, x\rangle \in \mathbb{B}^* \times X^*$ if on input p its output begins with the string x. A sequence of requests (request sequence) defines a process: the least process which satisfies all its requests. The processes generated by proper prefixes of an infinite request sequence are approximations to the process generated by the entire sequence. We will build our process by layering request upon request, forming a sequence of processes approximating the final process in the limit.

We next develop a method for building processes with particular semimeasures. Paralleling the above, a semimeasure ρ satisfies a pair $\langle \alpha, x \rangle \in \mathbb{Q} \times X^*$ if $\alpha < \rho(x)$ holds. A sequence of such pairs defines the least semimeasure satisfying all pairs in the sequence. The key idea behind our proof is that given a computable sequence of such pairs[7], we can construct a process whose semimeasure is the least semimeasure satisfying those pairs, subject to consistency constraints. An enumerable semimeasure ν is exactly a semimeasure for which the set of all pairs it satisfies can be enumerated. The least semimeasure satisfying all of these enumerated pairs is ν itself. This allows us to build a process implementing any enumerable semimeasure, completing the equivalence.

3.4.1 A process's semimeasure is enumerable

Lemma 3.14. For any process M the semimeasure μ_M is enumerable.

Proof. We show that the set

$$\{\langle \alpha, x \rangle \in \mathbb{Q} \times X^* : \alpha < \nu(x)\}$$

is computably enumerable. Given a pair $\langle \alpha, y \rangle$, Algorithm 3 will return 1 if and only if the pair is in the set. It computes an increasing approximation S of $\mu_M(y)$, effectively by simulating M on all possible inputs p.

As M is a process the following sets are computably enumerable:

1. $G_M = \{\langle p, x \rangle \in \mathbb{B}^* \times X^* : x \preceq M(p)\}$,
2. $H_M = \{\langle p, M(p) \rangle \in \mathbb{B}^* \times X^* :\ p \text{ is a non-terminal string of } M\}$.

A string p is *minimal* iff $p \in \min\{p \in \mathbb{B}^* :\ y \preceq M(p)\}$. A string p is added to S if and only if the conditional in step 5 succeeds and $S \leftarrow S + 2^{-|p|}$.

[7]Technically we will require α to be a dyadic rational, i.e. equal to $k2^{-d}$ for $k, d \in \mathbb{N}$. These are numbers which can be written with a finite number of binary digits, e.g. $21 \cdot 2^{-3} = 10.101$ in base 2. This is no problem as any real can be approximated arbitrarily closely by a dyadic rational.

Algorithm 3

```
 1: $S \leftarrow 0$, current lower bound of $\mu_M(y)$.
 2: $P \leftarrow \{\epsilon\}$, current set of programs.
 3: loop
 4:     Enumerate $G_M$ and $H_M$ simultaneously, yielding an element $\langle x, p\rangle$ of one.
 5:     if $p \in P$ and $y \preceq x$ then
 6:         $S \leftarrow S + 2^{-|p|}$
 7:         $P \leftarrow P \setminus \{p\}$
 8:     end if
 9:     if $p \in P$ and $\langle x, p\rangle \in H_M$ then
10:         $P \leftarrow P \cup \{p0, p1\} \setminus \{p\}$
11:     end if
12:     if $\alpha < S$ then
13:         Halt, returning 1.
14:     end if
15: end loop
```

Suppose p is minimal. Every proper prefix $p' \prec p$ has $M(p') \prec y$ and thus will not be removed from P in step 7. As every proper prefix of p is non-terminal, they are in H_M and will be successively generated and removed at step 10 starting from the prefix ϵ. Eventually the last proper prefix will be removed and p will be added. Following this p will be removed in step 7 and added to S.

Suppose p is not minimal. If $y \not\preceq M(p)$ then p will not be added to S (see step 5). Suppose $y \preceq M(p)$. Then there is a proper prefix $p' \prec p$ which is minimal. Since elements of P can be generated only by extending previous elements (see step 10), p' will be generated before p. However, since p' is minimal it will be added to S then removed from P. This means p will not be subsequently generated, and will not be added to S.

By the above, exactly the elements of the set

$$\min\{p \in \mathbb{B}^* : \ y \preceq M(p)\}$$

will be added to S. By Lemma 3.11 this shows the sum S converges to:

$$\sum_p 2^{-|p|} \, [p \in \min\{p \in \mathbb{B}^* : y \preceq M(p)\}] = \mu_M(y).$$

Since $\alpha < \mu_M(y)$ there will be a finite number of terms such that $S_n < \alpha$ where S_n denotes the value of variable S after n terms have been added to it. At this point the machine will return 1 in step 13. Conversely, if $\mu_M(y) \leq \alpha$ then $S_n \leq \alpha$ always holds, so Algorithm 3 will not terminate on input $\langle y, \alpha \rangle$. $\square$

3.4.2 Building processes

We first define valid request sequences r and the monotone functions $\mathcal{M}[r]$ generated by them. The definition of $\mathcal{M}[r]$ is implicit so we give two different explicit characterisations. These will pose useful in later proofs. We show if the request sequence r is computable then $\mathcal{M}[r]$ is a process, and that this method for defining processes is continuous in r. This property will allow us to build processes for any enumerable semimeasure in the next section. Finally, we prove an explicit characterisation for $\mu_{\mathcal{M}[r]}(x\downarrow)$ and explain why we didn't do so earlier.

The monotone function $\mathcal{M}[r]$ generated by a valid request sequence r

Not all request sequences can be satisfied. The request that $\Delta\Delta$ is output on input 0 and the request that $\Xi\Xi$ is output on input 00 cannot both be satisfied. We also wish to impose an ordering on requests: if we request that x is output on input p we cannot later have a request regarding a proper prefix of p. This ordering will mean as soon as we have a request involving p the output for all proper prefixes of p is fixed. We will see this is required for constructing processes.

This leads to the definition of a valid request sequence.

Definition 3.15. Let $r \in (\mathbb{B}^* \times X^*)^\#$ be a request sequence, and let $r_i = \langle p_i, x_i \rangle$ denote the ith element of r. This is a **valid request sequence** if for all $i, j \in \mathbb{N}$:

1. If $i < j$ and $p_i \preceq p_j$ then $x_i \preceq x_j$.
2. There is no $i < j$ with $p_i \succ p_j$.

$\square$

Recall that a monotone function $F \colon \mathbb{B}^\# \to X^\#$ is a function preserving prefixes: if $p \preceq q$ then $F(p) \preceq F(q)$. When recording the behaviour of a system this corresponds to temporal consistency: adding further inputs cannot change previous outputs. Processes are computable, continuous, monotone functions, but it is simpler to deal with continuous monotone functions in general until we introduce computability. We will turn out to get continuity for free, so we simplify matters by looking at monotone functions in general.

Definition 3.16. Let $r \in (\mathbb{B}^* \times X^*)^\#$ be a request sequence, and let $r_i = \langle p_i, x_i \rangle$ denote the ith element of r. A monotone function $F\colon \mathbb{B}^\# \to X^\#$ **satisfies** r if for every i

$$x_i \preceq F(p_i).$$

$\square$

Monotone functions can be partially ordered: $F \preceq G$ iff $F(p) \preceq G(p)$ for all $p \in \mathbb{B}^\#$. The set of all monotone functions satisfying a request sequence r has a least element under this order. This function outputs the least possible to satisfy the requests (as will be seen in Lemma 3.18). We will

Definition 3.17. Given a valid request sequence r, the monotone function **generated by** r

$$\mathcal{M}[r]\colon \mathbb{B}^\# \to X^\#$$

is the least monotone function which satisfies r i.e. $\mathcal{M}[r]$ satisfies r, and given any F which satisfies r we have

$$\mathcal{M}[r](p) \preceq F(p) \quad \text{for all } p \in \mathbb{B}^\#.$$

$\square$

Note that $\mathcal{M}[r]$, if it exists, is unique. If we had another monotone function F minimally satisfying r then we would have both $\mathcal{M}[r](p) \preceq F(p)$ and $F(p) \preceq \mathcal{M}[r](p)$ for all p. This implies $F(p) = \mathcal{M}[r](p)$ and thus $F = \mathcal{M}[r]$.

Two explicit forms for $\mathcal{M}[r]$

Definition 3.17 of $\mathcal{M}[r]$ is implicit. We give two explicit forms establishing existence as a corollary. The first shows $\mathcal{M}[r]$ outputs the least necessary to satisfy all the requests. The second gives an inductive definition of $\mathcal{M}[r]$ in terms of r, each step making the smallest change necessary to satisfy each new request. This latter form will be especially useful for proving properties of $\mathcal{M}[r]$.

Recall that the least upper bound of a subset $S \subseteq P$ of a partially ordered set, denoted $\sup S$ if it exists, satisfies $s \preceq \sup S$ for all $s \in S$ (is an upper bound of S), and for any other upper bound u satisfies $\sup S \preceq u$ (is the least upper bound). In the partial order $X^{\#}$ of sequences $\sup S$ is the shortest sequence for which every $s \in S$ is a prefix.

Lemma 3.18. The monotone function $\mathcal{M}[r]$ generated by a valid request sequence r exists and

$$\mathcal{M}[r](p) = \sup\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\}$$

where r_i is the ith request in r.

Proof. We first show the expression is well defined by showing the set

$$\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\}$$

is a linear order under $\preceq$ (as $X^{\#}$ is a complete partial order any linearly ordered subset has a least upper bound). Take any two distinct elements y_i and y_j of the above set, supposing without loss of generality that $i < j$. Since $q_i, q_j \preceq p$ we must have either $q_i \preceq q_j$ or vice versa. As r is a valid request sequence we have $q_i \preceq q_j$ (second condition of Definition 3.15), therefore $y_i \preceq y_j$ (first condition of Definition 3.15), and so the above set is a linear order.

We now show

$$F(p) = \sup\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\}$$

is monotone, satisfies r, and is least.

F is monotone since if $p_1 \preceq p_2$ then

$$\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p_1 \text{ for some } i\} \subseteq \{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p_2 \text{ for some } i\}$$

so an upper bound of the RHS is an upper bound of the LHS i.e. $F(p_1) \preceq F(p_2)$.

F satisfies r as for any $r_i = \langle q_i, y_i\rangle$ we have $y_i \preceq F(q_i)$ by definition.

Finally, suppose G is another monotone function satisfying r. Fix an input $p \in \mathbb{B}^{\#}$.

For any request $r_i = \langle q_i, y_i \rangle$ where $q_i \preceq p$ we have

$$y_i \preceq G(q_i) \preceq G(p)$$

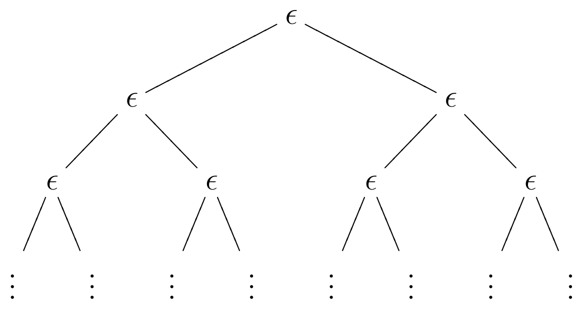
because G satisfies r and is monotone. But then

$$F(p) = \sup\{y_i : r_i = \langle q_i, y_i \rangle \text{ and } q_i \preceq p \text{ for some } i\} \preceq G(p)$$

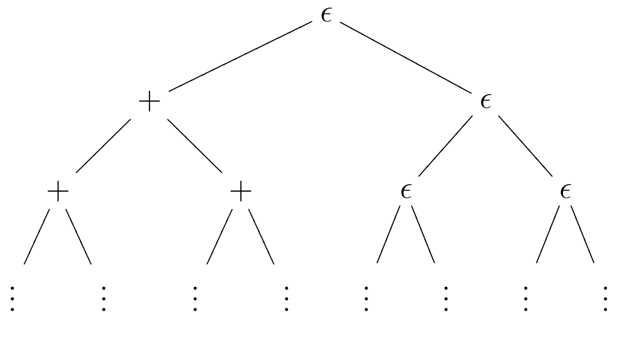
since $G(p)$ is an upper bound of this set. So F is least.

Therefore $F(p) = \mathcal{M}[r](p)$ for all p. □

We can visualise a monotone function $F \colon \mathbb{B}^\# \to X^\#$ as an infinite binary tree labelled with sequences over X. For example, the least such function, $\mathcal{M}[\epsilon](p) = \epsilon$, is



For finite request sequences r, the only difference between $\mathcal{M}[r]$ and $\mathcal{M}[r\langle p, x \rangle]$ is the subtree $p\,\mathbb{B}^\#$ has had its label changed to x. For instance, $\mathcal{M}[\langle 0, + \rangle]$ is



As a side note, if $r\langle p, x \rangle$ is a valid sequence then the subtree $p\,\mathbb{B}^\#$ has a constant value $y \in X^*$ in $\mathcal{M}[r]$ too, and $y \preceq x$ (this follows from Definition 3.15). That is, we only change constant-labelled subtrees, and we only add to their labels.

These observations are proved in general below.

Lemma 3.19. The monotone function $\mathcal{M}[r]$ generated by a valid request sequence r satisfies the identities:

1. $\mathcal{M}[\epsilon](p) = \epsilon$ for all $p \in \mathbb{B}^{\#}$.

2. If $r = r'\langle q, y\rangle$ for $\langle q, y\rangle \in \mathbb{B}^* \times X^*$, then

$$\mathcal{M}[r](p) = \begin{cases} y & \text{if } q \preceq p, \\ \mathcal{M}[r'](p) & \text{otherwise.} \end{cases}$$

3. For any request sequence r,

$$\mathcal{M}[r](p) = \sup\{\mathcal{M}[r'](p) : r' \text{ is finite and } r' \preceq r\}.$$

Proof. Invoking Lemma 3.18 we need only show that

$$\sup\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\}$$

satisfies these three identities.

For $r = \epsilon$ the RHS is the empty set, and $\sup \emptyset = \epsilon$.

For a finite valid sequence $r \neq \epsilon$ we have

$$r = r'\langle o, z\rangle$$

where $o \in \mathbb{B}^*$, $z \in X^*$, and r' is itself a valid sequence. Let

$$F(p) = \begin{cases} z & \text{if } o \preceq p, \\ \mathcal{M}[r'](p) & \text{otherwise.} \end{cases}$$

Fix an arbitrary $p \in \mathbb{B}^{\#}$. Now either $o \preceq p$ or not.

Suppose $o \not\preceq p$. Then,

$$\begin{aligned} & \{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\} \\ = \; & \{y_i : r'_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\} \end{aligned}$$

since $\langle o, z\rangle$, the only request in r but not r', is excluded by the condition $q_i \preceq p$. So,

$$F(p) = \mathcal{M}[r'](p) = \mathcal{M}[r](p)$$

where the first equality is by definition of F, the second by the above equality of sets.

Suppose $o \preceq p$. For any request $r'_i = \langle q_i, y_i\rangle$ in r' with $q_i \preceq p$ we have $q_i \preceq o$ as o and q_i are comparable (both are prefixes of p) and since r is a valid sequence we cannot have $q_i \succ o$ (condition 2 of Definition 3.15). By condition 1 of Definition 3.15 this means $y_i \preceq z$. Therefore

$$\sup\{y_i : r'_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\} \preceq z.$$

By this, and since $\langle o, z\rangle$ is the only request of r not in r', we have

$$\sup\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\} = z = F(p)$$

establishing the second identity.

For the final identity take a finite or infinite valid request sequence r. For any finite prefix r' of r we have

$$\mathcal{M}[r'](p) = \sup\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i \leq |r'|\}$$

Therefore

$$\begin{aligned} & \sup\{\mathcal{M}[r'](p) : r' \text{ is finite and } r' \preceq r\} \\ = \; & \sup\{y_i : r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p \text{ for some } i\} \\ = \; & \mathcal{M}[r](p). \end{aligned}$$

□

Computable request sequences generate processes

We show computable request sequences generate processes. The below construction actually works for invalid request sequences too, although the process generated will not equal the (undefined) $\mathcal{M}[r]$. This property will be used by Section 3.4.3 to show the set of all processes is computably enumerable.

Lemma 3.20. If the valid request sequence r is computable then $\mathcal{M}[r]$ is a process.

Proof. It's clear from the third identity of Lemma 3.19 that $\mathcal{M}[r]$ is continuous. It remains to show that the sets

$$\{\langle p, x\rangle \in \mathbb{B}^* \times X^* : x \preceq \mathcal{M}[r](p)\}$$

and

$$\{\langle p, M(p)\rangle \in \mathbb{B}^* \times X^* : \ p \text{ is a non-terminal string of } \mathcal{M}[r]\}$$

are computably enumerable.

Algorithm 4 given $p \in \mathbb{B}^*$ and $x \in X^*$ returns 1 if and only if $x \preceq \mathcal{M}[r](p)$, proving the first set computably enumerable.

Algorithm 4

```
1: $i \leftarrow 1$.
2: loop
3:     Compute $r_i = \langle q_i, y_i\rangle$, the $i$th request in $r$.
4:     if $q_i \preceq p$ and $x \preceq y_i$ then
5:         Return 1.
6:     end if
7:     $i \leftarrow i + 1$.
8: end loop
```

To see this, by Lemma 3.18 we know

$$\mathcal{M}[r](p) = \sup\{y_i : \text{whenever } r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p\}.$$

Because x is finite and the set of prefixes of $\mathcal{M}[r](p)$ form a linear order we have $x \preceq \mathcal{M}[r](p)$ if any only if $x \preceq y_i$ for some $r_i = \langle q_i, y_i\rangle$ with $q_i \preceq p$. Our algorithm halts in exactly this case.

Algorithm 5 given $p \in \mathbb{B}^*$ and $x \in X^*$ where p is nonterminal returns 1 if and only if $x = \mathcal{M}[r](p)$, proving the second set computably enumerable.

To prove correctness it suffices to show

$$\text{out} = \sup\{y_i : \text{whenever } r_i = \langle q_i, y_i\rangle \text{ and } q_i \preceq p\}$$

Algorithm 5

```
1: $i \leftarrow 1$.
2: out $\leftarrow \epsilon$.
3: **loop**
4:     Compute $r_i = \langle q_i, y_i \rangle$, the $i$th request in $r$.
5:     **if** $q_i \preceq p$ **then**
6:         out $\leftarrow y_i$
7:     **else if** $p \prec q_i$ **then**
8:         Return 1 if $x =$ out, 0 otherwise.
9:     **end if**
10:    $i \leftarrow i + 1$.
11: **end loop**
```

whenever $p \prec q_i$. We know $p \prec q_i$ necessarily holds for some i as p is nonterminal.

The variable out is nondecreasing, that if out has value o we only change it to value o' if $o \preceq o'$. Suppose $q_j \preceq p$ and $q_i \preceq p$ where $j < i$. Since r is a valid request sequence we cannot have $q_j \succ q_i$ so $q_j \preceq q_i$. But then $y_j \preceq y_i$, so step 6 never decreases out.

If $p \prec q_i$ holds then whenever $q_j \preceq p$ we have $j < i$. This holds because otherwise $q_i \succ q_j$ which contradicts r's validity as $i < j$. (We cannot have $i = j$ because q_i and q_j have different values.) This means there are a finite number of elements in the set

$$\{y_i : \text{whenever } r_i = \langle q_i, y_i \rangle \text{ and } q_i \preceq p\}$$

all of which are processed before statement 8 occurs. At this point out holds the value of the largest element, as it is nondecreasing, which is exactly

$$\sup\{y_i : \text{whenever } r_i = \langle q_i, y_i \rangle \text{ and } q_i \preceq p\}$$

proving correctness. □

Continuity of generated processes' semimeasures

We show the function $r \mapsto \mu_{\mathcal{M}[r]}$ is continuous in r. This allows us define an infinite request sequence r where $\mu_{\mathcal{M}[r^i]}$ tends to some limit for increasing finite prefixes $r^i \prec r$ and have $\mu_{\mathcal{M}[r]}$ *equal* that limit.

Lemma 3.21. If $r^i \in (\mathbb{B}^* \times X^*)^*$ is a sequence of valid finite request sequences with $r^i \preceq r^{i+1}$ for all i, then

$$\mu_{\mathcal{M}[r]}(x) = \lim_{i\to\infty} \mu_{\mathcal{M}[r^i]}(x)$$

for all $x \in X^*$, where

$$r = \sup\{r^i : i \in \mathbb{N}\}.$$

Proof.

$$\begin{aligned}
\lim_{i\to\infty} \mu_{\mathcal{M}[r^i]}(x) &\overset{(1)}{=} \lim_{i\to\infty} \lambda\{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r^i](p)\} \\
&\overset{(2)}{=} \lambda \bigcup_i \{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r^i](p)\} \\
&\overset{(3)}{=} \lambda\{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r^j](p) \text{ for some } j \in \mathbb{N}\} \\
&\overset{(4)}{=} \lambda\{p \in \mathbb{B}^\# : x \preceq \sup\{\mathcal{M}[r^j](p) : j \in \mathbb{N}\}\} \\
&\overset{(5)}{=} \lambda\{p \in \mathbb{B}^\# : x \preceq \sup\{\mathcal{M}[r'](p) : r' \text{ is finite and } r' \preceq r\}\} \\
&\overset{(6)}{=} \lambda\{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r](p)\} \\
&\overset{(7)}{=} \mu_{\mathcal{M}[r]}(x).
\end{aligned}$$

Steps 1 and 7 hold by definition of $\mathcal{M}[r^i]$ and $\mathcal{M}[r]$. We have

$$\{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r^i](p)\} \subseteq \{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r^{i+1}](p)\}$$

because $\mathcal{M}[r^i](p) \preceq \mathcal{M}[r^{i+1}](p)$ since $r^i \preceq r^{i+1}$. This implies step 2 as λ is a measure. Step 3 also holds by the above subset relation. Note that

$$x \preceq \sup\{\mathcal{M}[r^j](p) : j \in \mathbb{N}\} \quad \text{implies} \quad x \preceq \mathcal{M}[r^j](p) \text{ for some } j \in \mathbb{N}$$

as x is finite and the set of prefixes of $\sup\{\mathcal{M}[r^j](p) : j \in \mathbb{N}\}$ form a linear order. The converse holds by definition of sup. This implies step 4. For step 5 note that for any finite $r' \preceq r$ there exists an r^j such that $r' \preceq r^j$. This means that although

$$\{\mathcal{M}[r^j](p) : j \in \mathbb{N}\} \subseteq \{\mathcal{M}[r'](p) : r' \text{ is finite and } r' \preceq r\}$$

any upper bound for the left hand side must be an upper bound for the right hand side, so their least upper bounds are equal. Finally, step 6 holds by Lemma 3.19.

□

Explicit form for $\mu_{\mathcal{M}[r]}(x\downarrow)$

The following lemma and proof is similar to Lemma 3.11. The motivation is again computational, this lemma being applied in the following sections. We follow with a counterexample showing why this couldn't be proved with Lemma 3.11.

Lemma 3.22. If $r \in (\mathbb{B}^* \times X^*)^*$ is a valid finite request sequence then

$$\mu_{\mathcal{M}[r]}(x\downarrow) = \sum_p 2^{-|p|} \; [p \in \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r](p') \text{ for all } p' \succeq p\}]$$

for all $x \in X^*$.

Proof. We have

$$\begin{aligned}
\mu_{\mathcal{M}[r]}(x\downarrow) &= \mu_{\mathcal{M}[r]}(x) - \sum_{c \in X} \mu_{\mathcal{M}[r]}(xc) \\
&= \lambda\left(\{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r](p)\} \setminus \bigcup_{c \in X} \{p \in \mathbb{B}^\# : xc \preceq \mathcal{M}[r](p)\}\right) \\
&= \lambda(\{p \in \mathbb{B}^\# : x = \mathcal{M}[r](p)\}) \\
&= \lambda(\mathcal{M}[r]^{-1}(\{x\})).
\end{aligned}$$

The first step by Equation 2.1 defining $\nu(x\downarrow)$ for a semimeasure. The second by Definition 3.10 of a process's semimeasure, and as λ is a measure.

Suppose, now, we have an infinite sequence $p \in \mathbb{B}^\infty$ with $\mathcal{M}[r](p) = x$. Since r is finite in length, the maximum request program length

$$l_r = \max\{|q_i| : r_i = \langle q_i, y_i \rangle \text{ for some } i\}$$

is finite. It is clear by, e.g. induction on identities 1 and 2 of Lemma 3.19, that if $l_r \leq |q|$ then $\mathcal{M}[r](q') = \mathcal{M}[r](q)$ for all $q' \succeq q$. But then we must have $\mathcal{M}[r](q') = x$ for all $q' \succeq p_{1:l_r}$. Thus

$$p \in \Gamma_{p_{1:l_r}} \subseteq \mathcal{M}[r]^{-1}(\{x\}).$$

As a result

$$\mathcal{M}[r]^{-1}(\{x\}) = \bigcup_{p \in \mathbb{B}^*} \Gamma_p[p \in \min\{p \in \mathbb{B}^* : \Gamma_p \subseteq \mathcal{M}[r]^{-1}(\{x\})\}] \cup T$$

where $T \subseteq X^*$ since every infinite string is contained in a cylinder set (see Section 2.3.4 for a similar equality). All the above sets are pairwise disjoint since if neither $p \preceq q$ nor $q \preceq p$ then $\Gamma_p \cap \Gamma_q = \emptyset$, and T is disjoint from the rest by its definition. We then have

$$\begin{aligned}
\lambda(\mathcal{M}[r]^{-1}(\{x\})) &= \sum_p \lambda(\Gamma_p)[p \in \min\{p \in \mathbb{B}^* : \Gamma_p \subseteq \mathcal{M}[r]^{-1}(\{x\})\}] + \lambda(T) \\
&= \sum_p 2^{-|p|}[p \in \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r](p') \text{ for all } p' \succeq p\}] + \lambda(T) \\
&= \sum_p 2^{-|p|}[p \in \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r](p') \text{ for all } p' \succeq p\}].
\end{aligned}$$

The first step since λ is a measure, the second by definition of λ, the last since $\lambda(\{x\}) = 0$ for finite strings $x \in X^*$. □

The above lemma does not work for all semimeasures, which is why it was not proved along with Lemma 3.11. It doesn't even work for all enumerable semimeasures, which is why the request sequence must be finite. The following counterexample is adapted from [Pol04].

Example 3.23. Suppose we have a process which reads in 2 bits and outputs 1 if they equal 01. Otherwise it reads in 3 bits, outputting 1 if they equal 001. This repeats with the nth step reading n bits and outputting 1 if they equal 0^n1, reading a further $n+1$ bits if they aren't. Define $A_i \subseteq \mathbb{B}^\#$ to be the set of input sequences after which the process *does not* output 1 within i steps i.e. the set of inputs where the process has output nothing after i steps. We have

$$A_i = \mathbb{B}^\# \setminus \bigcup_{n=1}^{i} \bigcup_{y \in \mathbb{B}^{k_n}} \Gamma_{y0^n1}$$

where $k_1 = 0$ and $k_{n+1} = k_n + (n+1)$, so the set A_i is measurable. Since $A_{i+1} \subseteq A_i$ we can show

$$\lambda(A) = \prod_{k=2}^{\infty}(1 - 2^{-k}) = \exp\left[\sum_{k=2}^{\infty} \ln(1 - 2^{-k})\right]$$

where $A = \bigcap_i A_i$. From $\ln(u) \geq (u-1)/u$ we have $\sum_k \ln(1-2^{-k}) \geq -\sum_k 1/(2^k - 1)$. As $\sum_{k\geq 2} 1/(2^k - 1) < \ln 2$ we have

$$1/2 < \lambda(A).$$

This description can be formalised into a process, so by the proof of Theorem 3.29 below there is an infinite request sequence r^∞ constructing it. By this and the proof of Lemma 3.22 above we have

$$\mu_{\mathcal{M}[r^\infty]}(\epsilon\downarrow) = \lambda(A) > 1/2$$

But, since the process can always be made to output 1, there is no $p \in \mathbb{B}^*$ with $\Gamma_p \subseteq A$. So

$$\mu_{\mathcal{M}[r^\infty]}(\epsilon\downarrow) \neq 0 = \sum_p 2^{-|p|} \, [p \in \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r](p') \text{ for all } p' \succeq p\}].$$

$\square$

3.4.3 Enumerating processes

It is an elementary result that the computable functions can be enumerated [Odi89]: fixing countable domain Y and codomain X there is a computable function

$$u \colon \mathbb{N} \times Y \to X$$

where for every computable function $f \colon Y \to X$ has an index i such that

$$u(i, y) = f(y) \quad \text{for all } y \in Y.$$

Or, defining $u_i(y) = u(i, y)$, we say there is a computable sequence $u_i \colon Y \to X$ of all computable functions. We extend this result to processes. This will allow us to define universal processes in Chapter 4.

We first establish a converse to Lemma 3.20 above.

Lemma 3.24. Given a process M there exists a computable valid request sequence r such that

$$M = \mathcal{M}[r].$$

Proof. Algorithm 6 computes a monotone sequence r^i of request sequences. We show this defines an infinite computable request sequence

$$r = \sup\{r^i : i \in \mathbb{N}\}$$

such that $M = \mathcal{M}[r]$. Without loss of generality, we assume our enumerations of G_M and H_M enumerates each element infinitely often.

Algorithm 6 Computing the request sequence generating M

```
1: $i \leftarrow 0$.
2: $P_0 = \emptyset$. $P_i$ is the current set of programs we have determined the full output for.
3: $r^0 = \epsilon$. $r^i$ is the current request sequence.
4: loop
5:     Enumerate $G_M$ and $H_M$ simultaneously, yielding an element $\langle p_i, x_i \rangle$ of one.
6:     if $p_i \in \min(\mathbb{B}^* \setminus P_i)$ then
7:         if there is no request $\langle p_i, y \rangle$ in $r^i$ with $x_i \prec y$ then
8:             $r^{i+1} = r^i \langle p_i, x_i \rangle$.
9:         else
10:            $r^{i+1} = r^i$.
11:        end if
12:        if $\langle p_i, x_i \rangle \in H_M$ then
13:            $P_{i+1} = P_i \cup \{p_i\}$.
14:        else
15:            $P_{i+1} = P_i$.
16:        end if
17:    end if
18: end loop
```

Because P_i is closed under prefixes (steps 2, 6 and 13), if step 6 lets a string p_i pass it will never let a proper prefix $p_j \prec p_i$ pass later. This, combined with the condition in step 7, means the request sequence r^i is always valid. So $r = \sup\{r^i : i \in \mathbb{N}\}$ is valid. This means both $\mathcal{M}[r^{i+1}]$ and $\mathcal{M}[r]$ are well defined.

Because r^i is increasing in i, by definition of r, and as we only add requests that M

satisfies to r, we have

$$\mathcal{M}[r^i](p) \preceq \mathcal{M}[r^{i+1}](p) \preceq \mathcal{M}[r](p) \preceq M(p)$$

for all $p \in \mathbb{B}^*$ and $i \in \mathbb{N}$.

Observe two facts:

1. If $p \in \min(\mathbb{B}^* \setminus P_i)$ for some i then $\mathcal{M}[r](p) = M(p)$.

   To see this, suppose $p \in \min(\mathbb{B}^* \setminus P_i)$. Only step 13 can remove p from this set. If this occurs by step 12 we know that $M(p_j) = x_j$ where $p_j = p$ for some $j > i$. By steps 7 and 8 we know $x_j \preceq \mathcal{M}[r^{j+1}](p_j) \preceq \mathcal{M}[r](p_j)$. So $\mathcal{M}[r](p) = M(p)$. Otherwise for any $x \preceq M(p)$ we will eventually enumerate $\langle p, x\rangle$ in step 5. Then by steps 6 and 7 we will have $x \preceq \mathcal{M}[r](p)$. So $M(p) = \mathcal{M}[r](p)$.

2. If p is nonterminal, i.e. if $\langle p, M(p)\rangle \in H_M$, then $p \in P_i$ for some i.

   To see this, note that if ϵ is nonterminal then eventually $\langle \epsilon, M(\epsilon)\rangle$ is enumerated from H_M whilst $P_i = \emptyset$. This means $\epsilon \in P_{i+1}$. Otherwise if $p \neq \epsilon$ is nonterminal let p' be defined by $p'b = p$ for $b \in \mathbb{B}$. One can see that p' is nonterminal, so by inductive hypothesis $p' \in P_i$ for some i. Eventually $\langle p, M(p)\rangle$ is enumerated from H_M, at which point step 13 adds p into P_j.

We know that $\epsilon \in \min(\mathbb{B}^* \setminus P_0)$, so by fact 1 above we have $\mathcal{M}[r](\epsilon) = M(\epsilon)$.

Let $p \in \mathbb{B}^*$ by an arbitrary string not equal to ϵ.

If p is nonterminal let p' be defined by $p'b = p$ for $b \in \mathbb{B}$. We see p' is nonterminal, so $p' \in P_i$ for some i by fact 2, and thus $p = p'b \in \min(\mathbb{B}^* \setminus P_i)$. Thus $\mathcal{M}[r](p) = M(p)$ by fact 1.

Otherwise p is terminal. Let $p^* \preceq p$ be the smallest terminal prefix of p. We have $M(p^*) = M(p)$ as p^* is terminal. One can also see that $\mathcal{M}[r](p^*) = \mathcal{M}[r](p)$ as terminal strings are never added to the set P_i. So we need only establish that $\mathcal{M}[r](p^*) = M(p^*)$. If $p^* = \epsilon$ then $M(p^*) = \mathcal{M}[r](p^*)$ by previous result. Otherwise p' defined by $p'b = p^*$ for $b \in \mathbb{B}$ is nonterminal. By facts 1 and 2 we then know $p' \in P_i$ for some i. This means $p^* = p'b \in \min(\mathbb{B}^* \setminus P_i)$, and so $\mathcal{M}[r](p^*) = M(p^*)$

□

Theorem 3.25. There is a computable sequence $M_i\colon \mathbb{B}^\# \to X^\#$ for $i \in \mathbb{N}$ of all processes. That is, M_i is a process for all i, every process M is equal to M_i for some i, and we can compute M_i from i.

Proof. It is easy to computably enumerate the finite computable request sequences as all finite sequences are computable, and as $(\mathbb{B}^* \times X^*)^*$ is countable. To enumerate the infinite computable request sequence simply note that an infinite request sequence is simply a computable function

$$f\colon \mathbb{N} \to \mathbb{B}^* \times X^*$$

and all such functions can be enumerated. By combining these enumerations we get an enumeration

$$e\colon \mathbb{N} \to (\mathbb{B}^* \times X^*)^\#$$

of all computable request sequences. This includes *invalid* request sequences too.

Define M_i to be the process generated by Algorithms 4 and 5 from the proof of Lemma 3.20, where we define the computable request sequence r to be $e(i)$. Note that this is a process even if $e(i)$ is an invalid request sequence.

Finally, note that by Lemma 3.24 above, for every process M there is a computable request sequence r such that $\mathcal{M}[r] = M$. Since e is an enumeration of all such sequences, we have $e(i) = r$ for some i. Thus $M = M_i$ for some i. □

3.4.4 Approximating semimeasures with processes

Our plan is to take an enumerable semimeasure ν and construct a computable sequence of request sequences r^i such that

$$\lim_{i\to\infty} \mu_{\mathcal{M}[r^i]}(x) = \nu(x) \quad \text{for all } x \in \mathbb{B}^\#.$$

By Lemma 3.21 we will have

$$\mu_{\mathcal{M}[r]}(x) = \nu(x) \quad \text{for all } x \in \mathbb{B}^*$$

where $r = \sup\{r^i : i \in \mathbb{N}\}$. The request sequence r is computable, since the sequence r^i is computable, so $\mathcal{M}[r]$ defines a process by Lemma 3.20. Thus any enumerable semimeasure is effectively equal to some process's semimeasure.

Appending a request to increase the semimeasure

Suppose we are approximating an enumerable semimeasure ν from below. That is, we are constructing a sequence of semimeasures ν_i below ν

$$\nu_i(x) \leq \nu(x)$$

with $\nu_i(x) \leq \nu_{i+1}(x)$ for all i and x. That ν is enumerable means we occasionally discovers facts such as $0.25 \leq \nu(\Delta\Delta\Xi)$ and wish to define a ν_{i+1} capturing this i.e. where $0.25 \leq \nu_{i+1}(\Delta\Delta\Xi)$ and $\nu_i(x) \leq \nu_{i+1}(x)$ for all x. This sort of thing is necessary for the sequence ν_i to tend to ν.

We must ensure ν_{i+1} is the least semimeasure capturing what we know, as because we can never decrease the semimeasure we must never have $\nu(x) < \nu_{i+1}(x)$ for any x. However, we cannot simply define

$$\nu_{i+1}(x) = \begin{cases} 0.25 & \text{if } x = \Delta\Delta\Xi, \\ \nu_i(x) & \text{otherwise} \end{cases}$$

because this needn't be a semimeasure (although we can assume that $\nu_i(x) \leq 0.25$).

Let's be concrete: suppose the semimeasure ν_1 is defined as follows

x	ϵ	Δ	$\Delta\Delta$	$\Delta\Delta\Xi$	$\Delta\Xi$	otherwise
$\nu_1(x)$	1	0.5	0	0	0.25	0

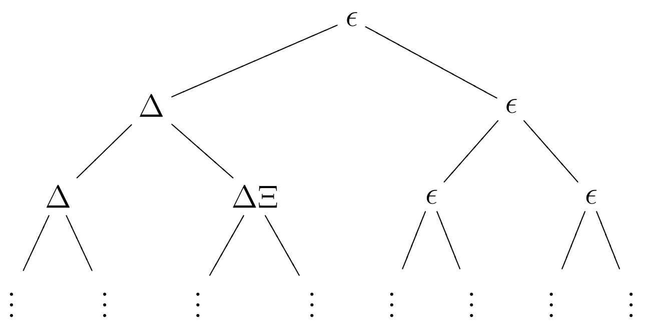
If we define ν_2 as above we would have $\nu_2(\Delta\Delta) < \nu_2(\Delta\Delta\Xi)$, an inconsistency. Instead, we define ν_2 to be the least semimeasure such that $\nu_1(x) \leq \nu_2(x)$ and $\nu_2(\Delta\Delta\Xi) = 0.25$. This semimeasure can be seen to be

x	ϵ	Δ	$\Delta\Delta$	$\Delta\Delta\Xi$	$\Delta\Xi$	otherwise
$\nu_2(x)$	1	0.5	0.25	0.25	0.25	0

As this is least we know $\nu_2(x) \leq \nu(x)$, so we can continue the sequence defining ν_3 as soon as we discover another fact about ν.

Since we wish to represent enumerable semimeasure by processes, all of our approximating semimeasures ν_i must be generated by processes $\mathcal{M}[r^i]$ with $r^i \preceq r^{i+1}$.

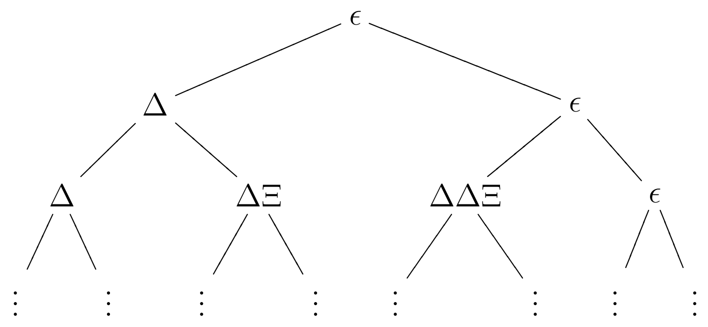
Suppose ν_1 is generated by the process



which is defined by the request sequence

$$\langle 0, \Delta\rangle\langle 01, \Delta\Xi\rangle.$$

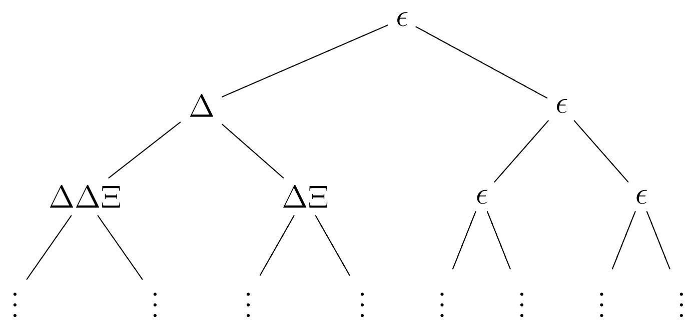
We wish to add a request $\langle p, \Delta\Delta\Xi\rangle$ for some $p \in \mathbb{B}^*$ to construct a process generating the least semimeasure ν_2. From Definition 3.15 and 3.19 one can see the value of the process must be constant on the subtree $p\,\mathbb{B}^\#$ for this to be a valid request sequence. We cannot, however, pick any such constant p. For example, if we use $p = 10$ we get



which has semimeasure

ϵ	Δ	$\Delta\Delta$	$\Delta\Xi$	$\Delta\Delta\Xi$	otherwise
1	0.75	0.25	0.25	0.25	0

This semimeasure assigns $\Delta\Delta\Xi$ the correct probability but assigns Δ too much. This occurs because there are nodes which have labels closer to $\Delta\Delta\Xi$ than node 01: node 00 has label Δ compared to node 10's ϵ. To increase the semimeasure minimally we must pick such a node. Using $p = 00$ gives us



which can be verified to have semimeasure ν_2.

Lemma 3.26 below proves the method used above works in general: if we increase the value of a constant node p to x, where node p has output closest to x, then we minimally increase the process's semimeasure's value at x by $2^{-|p|}$. This is our basic tool for increasing the measure assigned to a particular sequence. Later results will allow us to increase the semimeasure by arbitrary[8] amounts (Lemma 3.27), where possible (Lemma 3.28), allowing us to approximate any enumerable semimeasure by a process (Theorem 3.29).

Let a **finite process** be one generated by a finite request sequence. A program $q \in \mathbb{B}^*$ is **constant** in a process M if $M(q') = M(q)$ for all $q' \in \mathbb{B}^\#$ with $q' \succeq q$. This means all successors of q have the same output on M as q.

Lemma 3.26. Given a finite process $\mathcal{M}[r]$, a string $y \in X^*$, a program $q \in \mathbb{B}^*$ constant in $\mathcal{M}[r]$ with $\mathcal{M}[r](q) \prec y$, and if there is no p constant with $\mathcal{M}[r](q) \prec \mathcal{M}[r](p) \prec y$, then $\mu_{\mathcal{M}[r\langle q,y\rangle]}$ is the least semimeasure satisfying

1. $\mu_{\mathcal{M}[r]}(x) \leq \mu_{\mathcal{M}[r\langle q,y\rangle]}(x)$ for all $x \in X^*$,
2. $\mu_{\mathcal{M}[r]}(y) + 2^{-|q|} \leq \mu_{\mathcal{M}[r\langle q,y\rangle]}(y)$,

i.e. for any semimeasure ν, if $\mu_{\mathcal{M}[r]}(x) \leq \nu(x)$ for all x, and if $\mu_{\mathcal{M}[r]}(y)+2^{-|q|} \leq \nu(y)$, then $\mu_{\mathcal{M}[r\langle q,y\rangle]}(x) \leq \nu(x)$ for all x. Furthermore, we have equality:

$$\mu_{\mathcal{M}[r\langle q,y\rangle]}(y) = \mu_{\mathcal{M}[r]}(y) + 2^{-|q|}.$$

[8]Actually, only by dyadic amounts, but this is no problem since a sequence of dyadic rationals can have any real as its limit.

Proof. Since $r \preceq r\langle q,y\rangle$, and because $\mathcal{M}[r]$ is least by definition, we have $\mathcal{M}[r](p) \preceq \mathcal{M}[r\langle q,y\rangle](p)$ for all $p \in \mathbb{B}^\#$. But then

$$\{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r](p)\} \subseteq \{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r\langle q,y\rangle](p)\}$$

so by Definition 3.10 of a process's semimeasure:

$$\mu_{\mathcal{M}[r]}(x) \leq \mu_{\mathcal{M}[r\langle q,y\rangle]}(x) \quad \text{for all } x \in X^*.$$

By Lemma 3.19 we have

$$\mathcal{M}[r\langle q,y\rangle](p) = \begin{cases} y & \text{if } q \preceq p, \\ \mathcal{M}[r](p) & \text{otherwise.} \end{cases} \tag{3.1}$$

It follows that

$$\begin{aligned} \mu_{\mathcal{M}[r\langle q,y\rangle]}(y) &= \sum_p 2^{-|p|}\ [p \in \min\{p \in \mathbb{B}^* : y \preceq \mathcal{M}[r\langle q,y\rangle](p)\}] \\ &= \sum_p 2^{-|p|}\ [p \in \min\{p \in \mathbb{B}^* : y \preceq \mathcal{M}[r](p)\}] + 2^{-|q|} \\ &= \mu_{\mathcal{M}[r]}(y) + 2^{-|q|} \end{aligned}$$

by Lemma 3.11 and since q is the only additional minimal program whose output starts with y.

It remains to show $\mu_{\mathcal{M}[r\langle q,y\rangle]}$ is the least such semimeasure. Suppose ν is a semimeasure where $\mu_{\mathcal{M}[r]}(x) \leq \nu(x)$ for all x and where $\mu_{\mathcal{M}[r]}(y) + 2^{-|q|} \leq \nu(y)$. We prove $\mu_{\mathcal{M}[r\langle q,y\rangle]}(x) \leq \nu(x)$ by cases. Let $y_{\text{old}} = \mathcal{M}[r](q)$ and recall that $y = \mathcal{M}[r\langle q,y\rangle](q)$.

1. Not $y_{\text{old}} \prec x \preceq y$. By Equation 3.1 above, $\mathcal{M}[r](p)$ only differs from $\mathcal{M}[r\langle q,y\rangle](p)$ when $q \preceq p$. In this case $\mathcal{M}[r](p) = y_{\text{old}}$ and $\mathcal{M}[r\langle q,y\rangle](p) = y$. Consider the sets

$$\{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r\langle q,y\rangle](p)\} \quad \text{and} \quad \{p \in \mathbb{B}^\# : x \preceq \mathcal{M}[r](p)\}.$$

   Because $\mathcal{M}[r](p) \preceq \mathcal{M}[r\langle q,y\rangle](p)$, the only case where the inequality $x \preceq$

$\mathcal{M}[r\langle q,y\rangle](p)$ can hold without $x \preceq \mathcal{M}[r](p)$ also holding is if

$$\mathcal{M}[r](p) \prec x \preceq \mathcal{M}[r\langle q,y\rangle](p).$$

But this means $y_{\text{old}} \prec x \preceq y$. Since x is not between these values, by hypothesis, the sets are equal and so $\mu_{\mathcal{M}[r]}(x) = \mu_{\mathcal{M}[r\langle q,y\rangle]}(x)$ by definition. It follows that

$$\mu_{\mathcal{M}[r\langle q,y\rangle]}(x) = \mu_{\mathcal{M}[r]}(x) \leq \nu(x).$$

2. $y_{\text{old}} \prec x \preceq y$. We prove coherence by induction over all such x. For $x = y$ we have

$$\mu_{\mathcal{M}[r\langle q,y\rangle]}(y) = \mu_{\mathcal{M}[r]}(y) + 2^{-|q|} \leq \nu(y)$$

by hypothesis. Assume this holds for all z such that $x \prec z \preceq y$. Then,

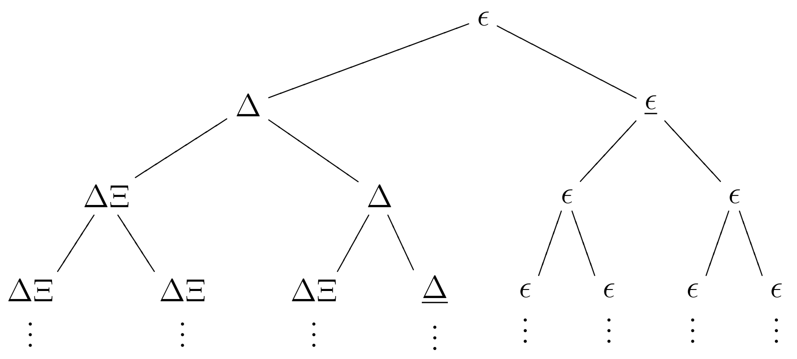
$$\begin{aligned}
\mu_{\mathcal{M}[r\langle q,y\rangle]}(x) &= \mu_{\mathcal{M}[r\langle q,y\rangle]}(x\downarrow) + \sum_{c\in X} \mu_{\mathcal{M}[r\langle q,y\rangle]}(xc) \\
&\leq \mu_{\mathcal{M}[r\langle q,y\rangle]}(x\downarrow) + \sum_{c\in X} \nu(xc) \\
&= \sum_{p} 2^{-|p|} \left[p \in \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r](p') \text{ for all } p' \succeq p\}\right] \\
&\quad + \sum_{c\in X} \nu(xc) \\
&= 0 + \sum_{c\in X} \nu(xc) \\
&\leq \nu(x).
\end{aligned}$$

The second step is by inductive hypothesis, the third step follows from Lemma 3.22, the fourth because by hypothesis there is no p constant in $\mathcal{M}[r]$ with $y_{\text{old}} \preceq \mathcal{M}[r](p) = x \prec y$.

□

Increasing a semimeasure by an arbitrary amount

Suppose we wish to allocate $\alpha = 0.5$ extra measure to the sequence $y = \Delta\Delta\Xi$ in the finite process $\mathcal{M}[r]$



We cannot simply apply Lemma 3.26 because there is no single program we can allocate with measure 0.5. We will show how Lemma 3.27 below constructs the process satisfying this allocation.

The precondition of Lemma 3.27 below is that we have sufficient free space to allocate the requested measure. That is,

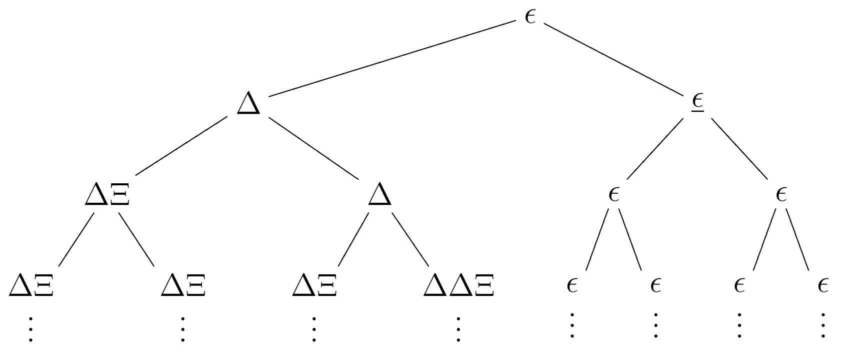
$$\begin{aligned}\alpha &\leq \sum_{x \prec y} \mu_{\mathcal{M}[r]}(x\downarrow) \\ &= \sum_{x \prec y} \sum_{p} 2^{-|p|} \, [p \in \min\{p : x = \mathcal{M}[r](p') \text{ for all } p' \succeq p\}].\end{aligned}$$

Notice the summation is over all minimal free programs Lemma 3.26 could be invoked with (although, importantly, we would have to use the programs with longer output first). In our example this evaluates to

$$0.5 \leq 2^{-|011|} + 2^{-|1|} = 0.75.$$

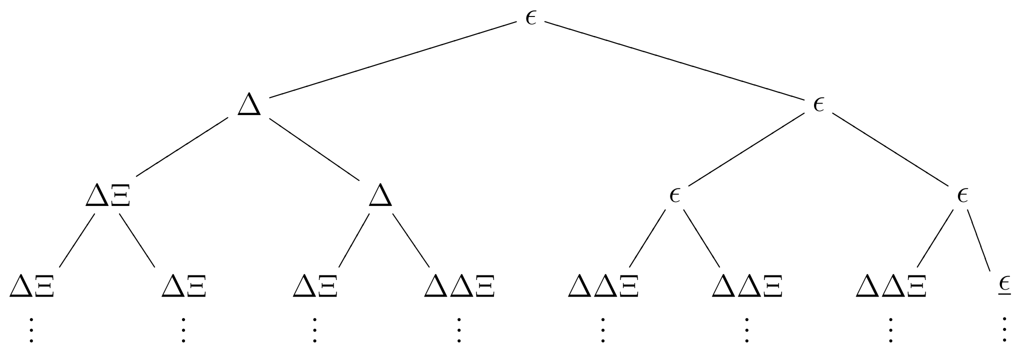
The minimal programs 011 and 1 form the set F_0 of minimal free programs at round 0 (see the proof below). In each diagram the elements of F_i will be underlined (note that F_0 is underlined in the previous diagram).

The algorithm first selects the program 011 as it is the minimal free program with greatest output. Allocating this to $\Delta\Delta\Xi$ gives us $\mathcal{M}[r^1]$


ϵ
Δ $\underline{\epsilon}$
$\Delta\Xi$ Δ ϵ ϵ
$\Delta\Xi$ $\Delta\Xi$ $\Delta\Xi$ $\Delta\Delta\Xi$ ϵ ϵ ϵ ϵ

This process has $1/8 = 0.125$ extra measure allocated to $\Delta\Delta\Xi$, there is 3/8 remaining.

The algorithm next selects the program 1. Since $3/8 < 1/2 = 2^{-|1|}$ the algorithm allocates not 1 but some children of 1 to $\Delta\Delta\Xi$, namely 100, 101 and 110. After three invocations of Lemma 3.26 this gives us $\mathcal{M}[r^2]$


ϵ
Δ ϵ
$\Delta\Xi$ Δ ϵ ϵ
$\Delta\Xi$ $\Delta\Xi$ $\Delta\Xi$ $\Delta\Delta\Xi$ $\Delta\Delta\Xi$ $\Delta\Delta\Xi$ $\Delta\Delta\Xi$ $\underline{\epsilon}$

As this now has 0.5 more measure assigned to $\Delta\Delta\Xi$ than the initial process we are complete: this is the process output by the lemma.

Recall that a **dyadic rational** α is one equal to $k2^{-d}$ for some $k, d \in \mathbb{N}$. These are exactly the rationals which can be written with a finite number of binary digits. Also, a finite process is one generated by a finite request sequence.

Lemma 3.27. Given a finite process $\mathcal{M}[r]$, a string $y \in X^*$, and a positive dyadic rational α such that

$$\alpha \leq \sum_{x \prec y} \mu_{\mathcal{M}[r]}(x\downarrow),$$

there effectively exists a finite request sequence $r' \succeq r$ such that $\mu_{\mathcal{M}[r']}$ is the least semimeasure satisfying

1. $\mu_{\mathcal{M}[r]}(x) \leq \mu_{\mathcal{M}[r']}(x)$ for all $x \in X^*$,
2. $\mu_{\mathcal{M}[r]}(y) + \alpha \leq \mu_{\mathcal{M}[r']}(y)$.

Furthermore, we have equality:

$$\mu_{\mathcal{M}[r']}(y) = \mu_{\mathcal{M}[r]}(y) + \alpha.$$

Proof. Algorithm 7 computes r' from r: when the algorithm terminates r' is defined to be r^i. It remains to prove the algorithm correct.

Algorithm 7 Allocate α extra measure to y in $\mathcal{M}[r]$

```
1: $i \leftarrow 0$.
2: $\alpha_0 = \alpha$.
3: $r^0 = r$.
4: **repeat**
5:     Select a node $q_i$ such that
           $$q_i \in \bigcup_{x \prec y} \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r^i](p') \text{ for all } p' \succeq p\}$$
       and where $\mathcal{M}[r^i](q_i)$ is maximal in the set (i.e. for all other $p$'s in the set, $\mathcal{M}[r^i](p) \preceq \mathcal{M}[r^i](q_i)$).
6:     **if** $2^{-|q_i|} < \alpha_i$ **then**
7:         Define $r^{i+1}$ from $r^i$ using Lemma 3.26 to change $q_i$'s output to $y$.
8:         $\alpha_{i+1} = \alpha_i - 2^{-|q_i|}$.
9:     **else**
10:        Let $k_i, d_i$ be natural numbers such that
               $$k_i \, 2^{-d_i} = \alpha_i$$
11:        Repeatedly using Lemma 3.26 change the output of the $k_i$ strings
               $$\{q_i b : \ b \in \mathbb{B}^{d_i - |q_i|} \text{ and } b < k_i \text{ interpreting } b \text{ as a binary number}\}$$
       to $y$ in $\mathcal{M}[r^i]$ forming $r^{i+1}$.
12:        $\alpha_{i+1} = 0$.
13:    **end if**
14:    $i \leftarrow i + 1$.
15: **until** $\alpha_i = 0$
```

Let

$$F_i = \bigcup_{x \prec y} \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r^i](p') \text{ for all } p' \succeq p\}$$

be the set of "free nodes" available to us at each step.

We first show we can finitely compute the (finite) set F_i for any i. This will mean step 5 succeeds so long as F_i is nonempty (which we also show). With this it is clear each step of the algorithm is finitely computable, so long as the applications of Lemma 3.26 in steps 7 and 11 are valid. We then establish the invariants

1. $\mu_{\mathcal{M}[r^i]}(y) = \mu_{\mathcal{M}[r]}(y) + (\alpha - \alpha_i)$.
2. For any semimeasure ν, if $\mu_{\mathcal{M}[r]}(x) \leq \nu(x)$ for all x, and if $\mu_{\mathcal{M}[r]}(y) + (\alpha - \alpha_i) \leq \nu(y)$, then $\mu_{\mathcal{M}[r^i]}(x) \leq \nu(x)$ for all x.
3. If $2^{-|q_i|} < \alpha_i$ then
$$F_{i+1} = F_i \setminus \{q_i\}.$$
4. $\alpha_i \leq \sum_{x \prec y} \mu_{\mathcal{M}[r^i]}(x{\downarrow})$.

for each value of i visited by the algorithm, whilst simultaneously showing each application of Lemma 3.26 in steps 7 and 11 is valid.

As the algorithm halts when $\alpha_i = 0$, the first two invariants show $r' = r^i$ satisfies the required conditions of the lemma. By Lemma 3.22 we have

$$\mu_{\mathcal{M}[r^i]}(x{\downarrow}) = \sum_p 2^{-|p|}\, [p \in \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r^i](p') \text{ for all } p' \succeq p\}].$$

This, by the last invariant and by definition of F_i, means F_i is nonempty whenever $\alpha_i > 0$, which means step 5 always succeeds in finding a q_i. This also shows the algorithm goes down the else branch of the if statement as soon as F_i has only one remaining member, after which it halts. As F_0 is a finite set (we show this below), the third invariant shows the algorithm terminates as F_i continuously decreases in size.

First observe that $\mathcal{M}[r^i](p)$ is finitely computable as r^i is finite: by Lemma 3.18

$$\mathcal{M}[r^i](p) = \sup\{y^i_j : r^i_j = \langle q^i_j, y^i_j\rangle \text{ for some } j \text{ and } q^i_j \preceq p\}$$

and the set on the right is finitely computable from r^i. If a program p has a proper prefix $p' \prec p$ with the same output $\mathcal{M}[r^i](p') = \mathcal{M}[r^i](p)$ it cannot be in

$$F_i = \bigcup_{x \prec y} \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r^i](p') \text{ for all } p' \succeq p\}$$

since it will not be minimal. From Lemma 3.19 it can be seen that $\mathcal{M}[r^i](p') \neq \mathcal{M}[r^i](p)$ for all proper prefixes $p' \prec p$ only if $\langle p, \mathcal{M}[r^i](p)\rangle$ is a request in r^i. Thus, computing the set F_i involves a finite search within r^i, so it is finitely computable. This means F_i is itself a finite set.

We establish the invariants by cases: either $2^{-|q_i|} < \alpha_i$ or not. All invariants but the third trivially hold for $i = 0$, so we will inductively prove then for $i+1$ from i. (The third invariant is not established by inductive proof.)

Suppose $2^{-|q_i|} < \alpha_i$. Note that the criteria used to select q_i from F_i exactly matches the preconditions for Lemma 3.26 in step 7.

1. We have

$$\begin{aligned}
\mu_{\mathcal{M}[r^{i+1}]}(y) &= \mu_{\mathcal{M}[r^i]}(y) + 2^{-|q_i|} \\
&= \mu_{\mathcal{M}[r]}(y) + (\alpha - \alpha_i) + 2^{-|q_i|} \\
&= \mu_{\mathcal{M}[r]}(y) + (\alpha - \alpha_{i+1})
\end{aligned}$$

   where the first step follows from the Lemma 3.26 application in step 7, the second by inductive hypothesis, and the last by definition of α_{i+1} in step 8.

2. Suppose ν satisfies $\mu_{\mathcal{M}[r]}(x) \leq \nu(x)$ for all x and $\mu_{\mathcal{M}[r]}(y) + (\alpha - \alpha_{i+1}) \leq \nu(y)$. Since

$$(\alpha - \alpha_i) + 2^{-|q_i|} = (\alpha - \alpha_{i+1})$$

   we have $\mu_{\mathcal{M}[r]}(y) + (\alpha - \alpha_i) \leq \nu(y)$ so by inductive hypothesis $\mu_{\mathcal{M}[r^i]}(x) \leq \nu(x)$ for all x. We also have

$$\begin{aligned}
\mu_{\mathcal{M}[r^i]}(y) + 2^{-|q_i|} &= \mu_{\mathcal{M}[r^{i+1}]}(y) \\
&= \mu_{\mathcal{M}[r]}(y) + (\alpha - \alpha_{i+1}) \\
&\leq \nu(y)
\end{aligned}$$

   where the first step follows by construction of r^{i+1}, the second by the previous invariant, and the last by hypothesis. Thus by Lemma 3.26 we have $\mu_{\mathcal{M}[r^{i+1}]}(x) \leq \nu(x)$ for all x, since $\mu_{\mathcal{M}[r^{i+1}]}(x)$ is the least such measure.

3. By the construction using Lemma 3.26 in step 7, and by Lemma 3.19 we have

$$\mathcal{M}[r^{i+1}](p) = \begin{cases} y & \text{if } q_i \preceq p, \\ \mathcal{M}[r^i](p) & \text{otherwise.} \end{cases}$$

As a result,

$$\begin{aligned} F_{i+1} &= \bigcup_{x \prec y} \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r^{i+1}](p') \text{ for all } p' \succeq p\} \\ &= \bigcup_{x \prec y} \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r^{i}](p') \text{ for all } p' \succeq p\} \setminus \{q_i\} \\ &= F_i \setminus \{q_i\}. \end{aligned}$$

To see this note that q_i is in F_i but not F_{i+1}, by choice of q_i and by the above expression for $\mathcal{M}[r^{i+1}]$. Any proper extension of q_i is in neither set as it cannot be minimal. Otherwise p satisfies $q_i \not\preceq p$ and so $\mathcal{M}[r^{i+1}](p) = \mathcal{M}[r^i](p)$. As a result F_i and F_{i+1} are equal but for q_i, establishing the central equality.

4. We have:

$$\begin{aligned} \sum_{x \prec y} \mu_{\mathcal{M}[r^i]}(x\downarrow) &= \sum_{x \prec y} \sum_{p} 2^{-|p|} \, [p \in \min\{p \in \mathbb{B}^* : x = \mathcal{M}[r^i](p') \text{ for all } p' \succeq p\}] \\ &= \sum_{p \in F_i} 2^{-|p|} \\ &= \sum_{p \in F_{i+1}} 2^{-|p|} + 2^{-|q_i|} \\ &= \sum_{x \prec y} \mu_{\mathcal{M}[r^{i+1}]}(x\downarrow) + 2^{-|q_i|} \end{aligned}$$

The first step by Lemma 3.22, the second by definition of F_i, the third by the previous invariant, and the last by Lemma 3.22 and the definition of F_{i+1}.

As $\alpha_{i+1} = \alpha_i - 2^{-|q_i|}$, and as

$$\alpha_i \leq \sum_{x \prec y} \mu_{\mathcal{M}[r^i]}(x\downarrow)$$

by inductive hypothesis, we have

$$\alpha_{i+1} \leq \sum_{x \prec y} \mu_{\mathcal{M}[r^{i+1}]}(x\downarrow).$$

Suppose $\alpha_i \leq 2^{-|q_i|}$. It can be shown that if we can apply Lemma 3.26 to a single string q in a process $\mathcal{M}[r]$ we can instead apply it repeatedly to a set of strings

$$\{qs : s \in S\}$$

where S is a prefix free subsets of $\mathbb{B}^*$, constructing a $\mathcal{M}[r^S]$ such that $\mu_{\mathcal{M}[r^S]}$ is the least semimeasure satisfying

1. $\mu_{\mathcal{M}[r]}(x) \leq \mu_{\mathcal{M}[r^S]}(x)$ for all $x \in X^*$,
2. $\mu_{\mathcal{M}[r]}(y) + 2^{-|q|} \sum_{s \in S} 2^{-|s|} \leq \mu_{\mathcal{M}[r^S]}(y)$,

with $r \preceq r^S$, and where we have equality in the second statment. The central idea is to show that

$$\mathcal{M}[r^S](p) = \begin{cases} y & \text{if } qs \preceq p \text{ for some } s \in S, \\ \mathcal{M}[r](p) & \text{otherwise,} \end{cases}$$

where $r^{\emptyset} = r$ and r^S is defined from $r^{S \setminus s}$ by applying Lemma 3.26 to $\mathcal{M}[r^{S \setminus s}]$ and the string qs for $s \in S$.

The proof that the first two invariants hold is then a simple extension of the proof for $2^{-|q_i|} < \alpha_i$ using this result with $q = q_i$ and

$$S = \{s : \; s \in \mathbb{B}^{d_i - |q_i|} \text{ and } s < k_i \text{ interpreting } s \text{ as a binary number}\}$$

instead of Lemma 3.26. This works because

$$\begin{aligned} \alpha_i - \alpha_{i+1} &= \alpha_i \\ &= k_i \, 2^{-d_i} \\ &= 2^{-|q_i|} \sum_{s \in S} 2^{-|s|}. \end{aligned}$$

Note invariants 3 and 4 are trivial in this case because $\alpha_i \leq 2^{-|q_i|}$ and $\alpha_{i+1} = 0$. $\square$

Increasing a semimeasure to meet any semimeasure above it

The following lemma shows we can apply Lemma 3.27 whenever $\mu_{\mathcal{M}[r]}(x) \leq \nu(x)$ for all x and $\mu_{\mathcal{M}[r]}(y) + \alpha \leq \nu(y)$. This means we can always bring $\mu_{\mathcal{M}[r]}$ up to meet any upper bound.

Lemma 3.28. Given a finite process M and a semimeasure ν, if for all $x \in X^*$

$$\mu_M(x) \leq \nu(x)$$

then for all $x \in X^*$

$$\nu(x) - \mu_M(x) \leq \sum_{y \,\prec\, x} \mu_M(y\downarrow).$$

Proof.

$$\begin{aligned}
\nu(x) - \mu_M(x) &\leq 1 - \sum_{i=1}^{|x|} \sum_{c \neq x_i} \nu(x_{1:i-1}c) - \mu_M(x) \\
&\leq 1 - \sum_{i=1}^{|x|} \sum_{c \neq x_i} \mu_M(x_{1:i-1}c) - \mu_M(x) \\
&= \sum_{i=1}^{|x|} \mu_M(x_{1:i-1}\downarrow) \\
&= \sum_{y \,\prec\, x} \mu_M(y\downarrow).
\end{aligned}$$

For any semimeasure ρ and any x by inductive application of

$$\rho(x_{1:i}) = \rho(x_{1:i-1}) - \rho(x_{1:i-1}\downarrow) - \sum_{c \neq x_i} \rho(x_{1:i-1}c)$$

(by Equation 2.1 restating semimeasure coherence) we have

$$\rho(x) = 1 - \sum_{i=1}^{|x|} \rho(x_{1:i-1}\downarrow) - \sum_{i=1}^{|x|} \sum_{c \neq x_i} \rho(x_{1:i-1}c)$$

as $\rho(\epsilon) = 1$ and $x_{1:0} = \epsilon$. The first and third steps apply this identity. The second applies the inequality $\mu_M(x) \leq \nu(x)$. The last equality holds as the strings $x_{1:i-1}$ for $i \leq |x|$ are all the strict prefixes of x. $\square$

3.4.5 Enumerable semimeasures are those generated by processes

We are finally ready to prove the desired result: enumerable semimeasures are exactly those generated by processes.

Theorem 3.29. For any enumerable semimeasure ν there effectively exists a process M where

$$\nu(x) = \mu_M(x) \quad \text{for all } x \in X^*.$$

Conversely, for any process M the semimeasure $\mu_M(x)$ is enumerable.

Proof. The converse is exactly Lemma 3.14 above.

Algorithm 8 computes a sequence r^i of finite request sequences such that

$$\lim_{i\to\infty} \mu_{\mathcal{M}[r^i]}(x) = \nu(x) \quad \text{for all } x \in \mathbb{B}^*.$$

By Lemma 3.21 we will have

$$\mu_{\mathcal{M}[r]}(x) = \nu(x) \quad \text{for all } x \in \mathbb{B}^*$$

where $r = \sup\{r^i : i \in \mathbb{N}\}$. The request sequence r is computable since the sequence r^i is computable, so $\mathcal{M}[r]$ defines a process by Lemma 3.20. Defining $M = \mathcal{M}[r]$ completes the theorem.

We first show we can compute $\mu_{\mathcal{M}[r^i]}(y_i)$, and that we always satisfy the preconditions of Lemma 3.27 in step 7. This shows the algorithm is well-defined. We end by showing $\mu_{\mathcal{M}[r^i]}(x)$ tends to $\nu(x)$.

First, observe that $\mathcal{M}[r](p)$ is finitely computable when r is finite: by Lemma 3.18

$$\mathcal{M}[r](p) = \sup\{y_i : r_i = \langle q_i, y_i\rangle \text{ for some } i \text{ and } q_i \preceq p\}$$

and the set on the right is finitely computable from r. By Lemma 3.11 we have

$$\mu_{\mathcal{M}[r]}(y) = \sum_p 2^{-|p|}\ [p \in \min\{p \in \mathbb{B}^* : y \preceq \mathcal{M}[r](p)\}].$$

If a program p has a proper prefix $p' \prec p$ with the same output $\mathcal{M}[r](p') = \mathcal{M}[r](p)$ it cannot be in the above minimal set. From Lemma 3.19 it can be seen that

Algorithm 8 Compute a sequence r^i such that $\lim_{i\to\infty}\mu_{\mathcal{M}[r^i]} = \nu$

```
1: $i \leftarrow 0$.
2: $r^0 = \epsilon$.
3: loop
4:     Take a dyadic rational $\alpha_i$ and a string $y_i$ from the enumeration of
           $\{\langle \alpha, y\rangle \in \mathbb{Q} \times X^* : \alpha < \nu(y)\}$.
5:     Compute the value of $\mu_{\mathcal{M}[r^i]}(y_i)$.
6:     if $\mu_{\mathcal{M}[r^i]}(y_i) < \alpha_i$ then
7:         Use Lemma 3.27 to allocate $\alpha_i - \mu_{\mathcal{M}[r^i]}(y_i)$ extra measure to $y_i$, defining
           the request sequence $r^{i+1}$ from $r^i$.
8:     else
9:         $r^{i+1} = r^i$.
10:    end if
11:    $i \leftarrow i + 1$.
12: end loop
```

$\mathcal{M}[r](p') \neq \mathcal{M}[r](p)$ for all proper prefixes $p' \prec p$ only if $\langle p, \mathcal{M}[r](p)\rangle$ is a request in r. Thus finding the set

$$\min\{p \in \mathbb{B}^* : y \preceq \mathcal{M}[r](p)\} \subseteq \{p \in \mathbb{B}^* : \mathcal{M}[r](p) \neq \mathcal{M}[r](p') \text{ for all } p' \prec p\}$$

involves a finite search through r. So $\mu_{\mathcal{M}[r]}(x)$ is finitely computable for finite r.

Next, we show the preconditions of Lemma 3.27 are always satisfied whilst establishing that

$$\mu_{\mathcal{M}[r^i]}(x) \leq \nu(x) \quad \text{for all } x \in X^*$$

holds for all i. It is clear that

$$\mu_{\mathcal{M}[r^0]}(x) = \mu_{\mathcal{M}[\epsilon]}(x) \leq \nu(x) \quad \text{for all } x \in X^*$$

since $\mu_{\mathcal{M}[\epsilon]}(x)$ is the least semimeasure. Suppose the above bound holds for $\mu_{\mathcal{M}[r^i]}$. If $\alpha_i \leq \mu_{\mathcal{M}[r^i]}(y_i)$ then $r^{i+1} = r^i$ so the bound trivially holds for $\mu_{\mathcal{M}[r^{i+1}]}$. Otherwise Lemma 3.28 implies that

$$\nu(y_i) - \mu_{\mathcal{M}[r^i]}(y_i) \leq \sum_{x \,\prec\, y_i} \mu_{\mathcal{M}[r^i]}(x\downarrow).$$

Since $\alpha_i < \nu(y_i)$ this means

$$\alpha_i - \mu_{\mathcal{M}[r^i]}(y_i) \leq \sum_{x \prec y_i} \mu_{\mathcal{M}[r^i]}(x\downarrow)$$

which allows us to apply Lemma 3.27. As a post condition we have

$$\mu_{\mathcal{M}[r^{i+1}]}(x) \leq \nu(x) \quad \text{for all } x \in X^*.$$

This shows our algorithm is well-defined.

As a second post condition of Lemma 3.27 we have,

$$\mu_{\mathcal{M}[r^{i+1}]}(y_i) = \mu_{\mathcal{M}[r^i]}(y_i) + (\alpha_i - \mu_{\mathcal{M}[r^i]}(y_i)) = \alpha_i$$

if $\mu_{\mathcal{M}[r^i]}(y_i) < \alpha_i$ and

$$\alpha_i \leq \mu_{\mathcal{M}[r^i]}(y_i)$$

otherwise. We will eventually enumerate α_i arbitrarily close to $\nu(y)$ for every $y \in \mathbb{B}^*$ since we enumerate every element of

$$\{\langle \alpha, y \rangle : \alpha \text{ is a dyadic rational, } y \in X^*, \text{ and } \alpha < \nu(y)\}$$

infinitely often (without loss of generality), and as the dyadic rationals are dense in the reals[9]. Thus,

$$\lim_{i\to\infty} \mu_{\mathcal{M}[r^i]}(x) = \nu(x) \quad \text{for all } x \in \mathbb{B}^*.$$

□

3.4.6 Interpreting the equivalence

We describe two interpretations of Theorem 3.29, the equivalence between enumerable semimeasures and processes. First is that any enumerable semimeasure corresponds to a probabilistic computable model of the sequence's generation. Second is that constructing an enumerable semimeasure's process can be seen as extracting the knowledge out of the semimeasure[10].

[9]This means we can approximate a real number arbitrarily closely by a dyadic rational.
[10]This interpretation suggested by Eliezer Yudkowsky in personal communication.

Uncertain knowledge corresponds to probabilistic models

A process generates an output sequence from an input sequence. Whenever

$$\nu(x) = \mu_M(x)$$

holds, the knowledge of the sequence embodied in ν is equivalent to (i.e. assigns the same probabilities as) the knowledge that the sequence is generated by the process M from an unbounded but completely unknown sequence (see also Section 3.2.3). This is a probabilistic (or stochastic) model of the sequence's generation. Since processes can be seen as computer programs with input and output (Section 3.2.2), the semimeasure $\mu_M(x)$ can be seen as describing the expected output of the program M with access to a purely random input stream.

For example, consider the sequence of floors an elevator visits, sampled at a fixed time interval. In a 5 floor building this is a sequence over the set $\{1, 2, 3, 4, 5\}$, such as $111112234554\cdots$. Suppose our knowledge about this sequence is described by the semimeasure ν. This knowledge may come from a simulation which models the people (randomly) arriving at floors and pressing buttons, and the elevator's response to these instructions. Using our knowledge, we can write a computer program which simulates the elevator and outputs the sequence of floors it visits. Since we are unsure of how many people will arrive when, the program uses the random noise input to make stochastic choices. This is a probabilistic computable model of the sequence's generation.

Extracting knowledge, or factoring out certainty

For every enumerable semimeasure ν there exists a process M such that

$$\nu(x) = \mu_M(x), \quad \text{for all } x \in X^*.$$

By Definition 3.10 we have

$$\nu(x) = \lambda(M^{-1}(\Gamma_x)).$$

The semimeasure ν represents partial knowledge about a sequence. Since λ represents complete ignorance about an unbounded sequence, and we know the process

M with complete certainty (i.e. we know which function $M\colon \mathbb{B}^{\#} \to X^{\#}$ it is), the above equality is a factoring of the knowledge represented by ν. The construction in Theorem 3.29 extracts the knowledge (or certainty) within ν into a process M, leaving behind complete ignorance (or uncertainty) within λ.

Chapter 4

Universal semimeasures

Semimeasures capture[1] uncertain knowledge about a sequence, so their accuracy is important. We evaluate a semimeasure's error[2] by comparing its predictions with reality. We measure the error (or total error) of a semimeasure ν by the negative logarithm

$$\mathrm{Err}[\nu(x\downarrow)] = -\log \nu(x\downarrow)$$

of the probability $\nu(x\downarrow)$ it assigns the sequence $x \in X^{\#}$ which actually occurs (see Section 1.2 for a derivation of this measure). The higher this probability the lower its error. A semimeasure's error is zero when it assigns probability one to the true sequence; the larger the error the more it deviates from this ideal. Error is called surprisal or self-information in information theory, and expected error is entropy [CT91].

Partial errors

$$\mathrm{Err}[\nu(x)] = -\log \nu(x)$$

for $x \in X^*$ measure the error when only a prefix $x \in X^*$ of the total sequence has been read. Partial errors can be used to measure the error rate, particularly useful if the total error $\mathrm{Err}[\nu(x\downarrow)]$ is infinite.

As a semimeasure's error depends on an unknown sequence it is natural to study the semimeasure's worst-case performance. If a semimeasure ν's error is at most

[1]Semimeasure *capture* the knowledge because probabilities are sufficient to make decisions [Ber93]

[2]Section 1.2 derived a measure of accuracy on the scale $[-\infty, 0]$, then negated it to form a nonnegative measure of inaccuracy or error.

a constant larger than another ρ's then we say ν **dominates** ρ (Section 4.1). If ν doesn't dominate ρ then ρ can arbitrarily outperform ν. Ideally we'd like to exclude such semimeasures.

For the class of enumerable semimeasures we can fulfill this goal: there exists enumerable semimeasures which dominate all other enumerable semimeasures. The previous chapter described semimeasures generated by processes. A process is universal if it simulates all others. The semimeasure μ_U of a universal process U dominates all enumerable semimeasures (Section 4.2).

In the definition of dominance there is a hidden additive constant: the maximum amount the dominated semimeasure can outperform the dominating one. The smaller this constant the better the worst-case performance. The semimeasure μ_U dominates any enumerable semimeasure ν with constant equal to ν's complexity relative to U (Sections 4.2.2, 4.2.4). For nice universal processes U there is reason to expect these constants to be small (Section 4.2.5), so for such U the semimeasure μ_U serves as a powerful predictor.

4.1 Dominance

Consider the semimeasures

$$\begin{aligned}
\nu_1(x) &= \begin{cases} 1 & \text{if } x = \epsilon, \\ 2^{-|x|-10} & \text{otherwise} \end{cases} \\
\nu_2(x) &= 2^{-2|x|} \\
\nu_3(x) &= \begin{cases} 1 & \text{if } x = \epsilon, \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

over the set $X = \{\Delta, \Xi\}$. If we thought the sequence was generated by first flipping a coin 10 times, then if we got 10 heads continuously flipping the coin to generate a sequence, we would use the semimeasure ν_1 to describe our knowledge. If we thought it were generated by continuously flipping two coins, where the first must be heads for the sequence to continue and the second selects the elements of the sequence, we would use the semimeasure ν_2. If we thought the sequence was definitely empty we would use the semimeasure ν_3.

The semimeasure ν_1 has the partial error $\text{Err}[\nu_1(x)] = |x| + 10$ for $x \neq \epsilon$ and $\text{Err}[\nu_1(\epsilon)] = 0$, whilst ν_2 has the partial error $\text{Err}[\nu_2(x)] = 2|x|$. Even though $\text{Err}[\nu_1(\Delta)] = 11 > 2 = \text{Err}[\nu_2(\Delta)]$ we have

$$\text{Err}[\nu_1(x)] \leq \text{Err}[\nu_2(x)] + 10$$

for all $x \in X^*$: ν_1's partial error is at most 10 bits larger than ν_2's. No such bound exists in the opposite direction as ν_2's error grows twice as fast as ν_1's. We say ν_1 **weakly dominates** ν_2 with constant 10.

For the semimeasures ν_2 and ν_3 we have

$$\text{Err}[\nu_2(x)] = 2|x| \leq \left\{\begin{matrix} 0 & \text{if } x = \epsilon \\ \infty & \text{otherwise} \end{matrix}\right\} = \text{Err}[\nu_3(x)]$$

and

$$\text{Err}[\nu_2(x{\downarrow})] = 2|z| + 1 \leq \left\{\begin{matrix} 0 & \text{if } z = \epsilon \\ \infty & \text{otherwise} \end{matrix}\right\} + 1 = \text{Err}[\nu_3(x{\downarrow})] + 1$$

for all $x \in X^*$. We say ν_2 **strongly dominates** ν_3 with constant 1.

Note that the semimeasure ν_1 has the total error $\text{Err}[\nu_1(x{\downarrow})] = \infty$ for $x \neq \epsilon$, whilst ν_2 has the total error $\text{Err}[\nu_2(x{\downarrow})] = 2|x| + 1$. In case there is no c such that

$$\text{Err}[\nu_1(x{\downarrow})] \leq \text{Err}[\nu_2(x{\downarrow})] + c$$

holds for all $x \in X^*$. So although ν_1 weakly dominates ν_2 it does not strongly dominate it.

Definition 4.1. A semimeasure ρ **weakly dominates** (or **dominates**) a semimeasure ν with constant $c \in \mathbb{R}$ if

$$\text{Err}[\rho(x)] \leq \text{Err}[\nu(x)] + c \quad \text{for all } x \in X^*.$$

If additionally

$$\text{Err}[\rho(x{\downarrow})] \leq \text{Err}[\nu(x{\downarrow})] + c \quad \text{for all } x \in X^*$$

then ρ **strongly dominates** ν with constant c. □

Dominance is a worst-case bound on error. Weak dominance bounds the total error

$\mathrm{Err}[\rho(\omega)]$ for infinite sequences $\omega \in X^\infty$, since if ρ weakly dominates ν with constant c then $\nu(\omega{\downarrow}) \leq 2^c \rho(\omega{\downarrow})$ for all infinite sequences $\omega \in X^\infty$ by Lemma 4.2 below and Equation (2.1), and so

$$\mathrm{Err}[\rho(\omega{\downarrow})] \leq \mathrm{Err}[\nu(\omega{\downarrow})] + c \quad \text{for all } \omega \in X^\infty.$$

Weak dominance also gives a sharp bound on the error rate

$$\lim_{n\to\infty} \mathrm{Err}[\rho(\omega_{1:n})]/n \leq \lim_{n\to\infty} \mathrm{Err}[\nu(\omega_{1:n})]/n$$

for all infinite sequences ω. Strong dominance additionally bounds the total error for finite sequences (Section 4.1.1 below shows weak dominance isn't sufficient).

Weak dominance is called simply dominance in the literature [LV97] [Hut05] and is normally and equivalently defined as in Lemma 4.2 below. Strong dominance appears to be novel.

Lemma 4.2. A semimeasure ρ weakly dominates a semimeasure ν if there exists a constant $k \in \mathbb{R}$ such that

$$\nu(x) \leq k\rho(x) \quad \text{for all } x \in X^*.$$

If additionally

$$\nu(x{\downarrow}) \leq k\rho(x{\downarrow}) \quad \text{for all } x \in X^*$$

then ρ strongly dominates ν.

Proof. Combine Definition 4.1 of dominance with Definition 1.3 of error. □

4.1.1 Weak and strong dominance are distinct

We have shown above that if ρ weakly dominates ν with constant c then

$$\mathrm{Err}[\rho(\omega{\downarrow})] \leq \mathrm{Err}[\nu(\omega{\downarrow})] + c \quad \text{for all infinite sequences } \omega \in X^\infty.$$

As hinted by the example of ν_1 and ν_2 in the previous section, we cannot extend this result to finite sequences $z \in X^*$. The following construction clarifies why this is so.

Example 4.3. Suppose ρ strongly dominates ν and $\nu(\epsilon\downarrow)$ is nonzero. Consider the semimeasure

$$\hat{\rho}(x) = \begin{cases} 1 & \text{if } x = \epsilon, \\ 1/|X| & \text{if } |x| = 1, \\ \rho(x)/|X| & \text{otherwise.} \end{cases}$$

Suppose $\rho = \mu_M$ for a process M i.e. that ρ represents the knowledge that the sequence was generated by M fed random input (always possible by Theorem 3.29). Then $\hat{\rho} = \mu_{\hat{M}}$ where $\hat{M}$ first randomly guesses a character, each character with equal probability $1/|X|$, outputs it, then runs M. If M's output agrees with $\hat{M}$'s guess then $\hat{M}$ outputs the rest of the sequence generated by M. Otherwise it outputs nothing more.

The semimeasure $\hat{\rho}$ dominates ν, since $\rho(x) \leq |X|\,\hat{\rho}(x)$. However, as $\nu(\epsilon\downarrow)$ is nonzero and $\hat{\rho}(\epsilon\downarrow)$ is zero, $\hat{\rho}$ does not strongly dominate ν. The reason behind this lack of strong dominance is $\hat{M}$ always outputs a first character where as, by hypothesis, the sequence described by ν can be empty.

One can generalise this. Suppose ρ strongly dominates ν and $\nu(\hat{x}\downarrow)$ is nonzero for some $\hat{x} \in X^*$. Then

$$\hat{\rho}(x) = \begin{cases} \rho(\hat{x})/|X| & \text{if } \hat{x} \prec x \text{ and } (|x| - |\hat{x}|) = 1, \\ \rho(x)/|X| & \text{if } \hat{x} \prec x \text{ and } (|x| - |\hat{x}|) > 1, \\ \rho(x) & \text{otherwise,} \end{cases}$$

corresponds to the machine which runs M and only tries to guess M's next character once M has output the string $\hat{x}$ (with $\hat{x} = \epsilon$ we get the previous construction). As before it will output its guess and only continue if it guesses correctly.

Again, $\hat{\rho}$ dominates ν since $\rho(x) \leq |X|\,\hat{\rho}(x)$. However, as $\nu(\hat{x}\downarrow)$ is nonzero and $\hat{\rho}(\hat{x}\downarrow)$ is zero $\hat{\rho}$ does not strongly dominate ν. □

As with weak and strong dominance above, later sections will introduce several concepts with weak and strong forms. These all stem from the difference between weak dominance which bounds partial errors

$$\mathrm{Err}[\rho(x)] \leq \mathrm{Err}[\nu(x)] + k \quad \text{for all } x \in X^*,$$

and strong dominance which bounds both partial and total errors

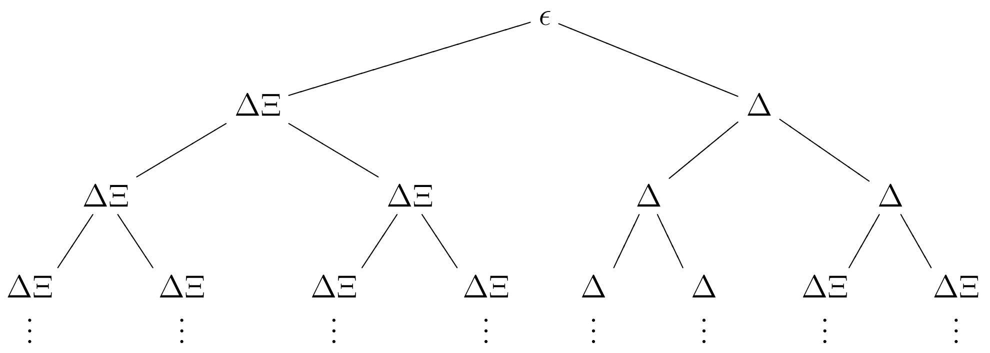
$$\mathrm{Err}[\rho(x\downarrow)] \leq \mathrm{Err}[\nu(x\downarrow)] + k \quad \text{for all } x \in X^*.$$

We will introduce the concepts of simulation, simulation complexity, universal processes, and universal semimeasures. Each of these will have a weak and a strong form. Just as strong dominance implies weak dominance, so too will the stronger concepts imply their weaker counterparts.

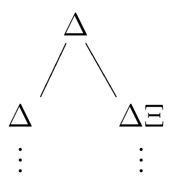
4.2 Universal semimeasures

4.2.1 Simulation

Consider the processes U



and M



The function

$$f(p) = \begin{cases} 1 & \text{if } p = \epsilon, \\ 10q & \text{if } p = 0q \text{ for } q \in \mathbb{B}^{\#}, \\ 110q & \text{if } p = 1q \text{ for } q \in \mathbb{B}^{\#}, \end{cases}$$

maps M's nodes into U's. This function has four properties

1. It preserves the structure of the tree: if $p \preceq q$ then $f(p) \preceq f(q)$. Formally, f is monotone.
2. It never maps different nodes onto the same node: if $p \neq q$ then $f(p) \neq f(q)$. Formally, f is injective.
3. It preserves labels: $M(p) = U(f(p))$.
4. It maps nodes at most a constant distance deeper down: $|f(p)| \leq |p| + 2$.

We will call such functions simulations (specifically, strong simulations). Simulations are interesting because their existence establishes dominance relations (Section 4.2.3).

The function $f_w(p) = 0p$ also satisfies the above properties if we weaken the second to $M(p) \preceq U(f_w(p))$. This allows a reduction of the constant in the fourth: $|f_w(p)| \leq |p| + 1$. We call such functions weak simulations.

Intuitively, a process U simulates another M if we can uniquely recode all of M inputs into inputs for U, without increasing their length more than a fixed amount. After this recoding, for a strong simulation U must output exactly what M would, but for a weak simulation U can output additional trailing symbols.

Definition 4.4. A process U **weakly simulates** another M with constant $k \in \mathbb{N}$ if there exists a monotone injective function $f \colon \mathbb{B}^\# \to \mathbb{B}^\#$ such that for all $p \in \mathbb{B}^\#$

1. $M(p) \preceq U(f(p))$,
2. $|f(p)| \leq |p| + k$.

If additionally $M(p) = U(f(p))$ for all p then M **strongly simulates** N. $\square$

Just as with weak and strong dominance, a process U can weakly simulate another process M for some constant but not strongly simulate it for any constant. For example, this happens with U defined by

$$\begin{tikzcd}
 & \Delta\Xi \arrow[dl, dash] \arrow[dr, dash] & \\
\Delta\Xi & & \Delta\Xi \\
\vdots & & \vdots
\end{tikzcd}$$

and M defined by

$$\begin{tikzcd}[column sep=small] & \Delta \arrow[dl, dash] \arrow[dr, dash] & \\ \Delta & & \Delta\Xi \\ \vdots & & \vdots \end{tikzcd}$$

4.2.2 Simulation complexity

We will find (Theorem 4.8 below) that a semimeasure μ_U dominates another μ_M with constant k whenever U simulates M with constant k. Since we want the sharpest bound it is useful to have notation for the least simulation constant. This measures the penalty incurred by simulating M on U, or the complexity of M relative to U.

Definition 4.5. The **weak simulation complexity** of a process M relative to another U is

$$\mathrm{S_U(M)} = \min\{\mathrm{k} : \ U \text{ weakly simulates } M \text{ with constant } k\}.$$

We define $\mathrm{S_U(M)} = \infty$ if U cannot weakly simulate M.

The **strong simulation complexity** of a process M relative to another U is

$$\mathrm{S_U(M{\downarrow})} = \min\{\mathrm{k} : \ U \text{ strongly simulates } M \text{ with constant } k\}.$$

We define $\mathrm{S_U(M{\downarrow})} = \infty$ if U cannot strongly simulate M. □

As all strong simulations are weak simulations we have

$$\mathrm{S_U(M)} \leq \mathrm{S_U(M{\downarrow})}$$

for all processes M and N.

Complexity of a semimeasure

By Theorem 3.29, every enumerable semimeasure is the semimeasure of some process. With this result we can naturally define the simulation complexity of a semimeasure to be the complexity of the simplest process which generates it. This will allow us to extend the dominance established below (Section 4.2.3) to all enumerable semimeasures.

Definition 4.6. The **simulation complexity** of an enumerable semimeasure ν relative to U is

$$\begin{aligned} \mathrm{S_U(\nu)} &= \min\{\mathrm{S_U(M)} : M \text{ is a process and } \mu_M(x) = \nu(x) \text{ for all } x \in X^*\} \\ \mathrm{S_U(\nu{\downarrow})} &= \min\{\mathrm{S_U(M{\downarrow})} : M \text{ is a process and } \mu_M(x) = \nu(x) \text{ for all } x \in X^*\} \end{aligned}$$

where the first is weak, the second strong. □

Complexity of a sequence

In a similar fashion to the above, we can define the simulation complexity of a sequence to be the simulation complexity of the simplest process which generates it. For a sequence $x \in X^{\#}$ denote by D_x the constant process defined by

$$D_x(p) = x, \quad \text{for all } p \in \mathbb{B}^{\#}.$$

A constant process outputs a fixed sequence regardless of its input. The simulation complexity of the process D_x relative to a process U is the natural way to define the complexity of the sequence x relative to U.

Definition 4.7. The **simulation complexity** of a sequence $x \in X^{\#}$ relative to U is

$$\begin{aligned} \mathrm{S_U(x)} &= \mathrm{S_U(D_x)} \\ \mathrm{S_U(x{\downarrow})} &= \mathrm{S_U(D_x{\downarrow})} \end{aligned}$$

where the first is weak, the second strong. □

The weak simulation complexity is exactly the monotone complexity [LV97] $Km(x)$ of x:

$$\begin{aligned} \mathrm{S_U(x)} &= \mathrm{S_U(D_x)} \\ &= \min\{k : \text{exists monotone injective } f \text{ where} \\ &\qquad\qquad x \preceq U(f(p)) \text{ and } |f(p)| \le |p| + k \text{ for all } p\} \\ &= \min\{|p| : x \preceq U(p)\} \\ &= Km(x) \end{aligned}$$

where the second step holds by using the mapping $f \mapsto f(\epsilon)$ to get a p such that $x \preceq U(p)$ and $|p| \leq k$, and because whenever $x \preceq U(p)$ the function $f : q \mapsto pq$ is a weak simulation of constant $|p|$ by the monotonicity of U.

The strong simulation complexity of the sequence x is a new complexity. We find

$$\begin{aligned} \mathrm{S_U(x{\downarrow})} &= \mathrm{S_U(D_x{\downarrow})} \\ &= \min\{k : \text{exists monotone injective } f \text{ where} \\ &\qquad\qquad x = U(f(p)) \text{ and } |f(p)| \leq |p| + k \text{ for all } p\} \\ &\leq \min\{|p| : x = U(p') \text{ for all } p' \succeq p\}. \end{aligned}$$

The inequality in the last step is necessary, as we can have a strong simulation f of constant k without having a constant subtree p with depth at most k. To see this, fix a sequence $\hat{x} \in X^{\#}$ and define the process U_n by

$$U_n(p) = \begin{cases} \hat{x}c & \text{if } p = b^n 0 p' \text{ for } b^n \in \mathbb{B}^n \text{ and } p' \in \mathbb{B}^{\#}, \\ \hat{x} & \text{otherwise,} \end{cases}$$

for some element $c \in X$. There is a strong simulation of $D_{\hat{x}}$ on U_n with constant 1:

$$f(p) = \begin{cases} b^n 1 p' & \text{if } p = b^n p' \text{ for } b^n \in \mathbb{B}^n \text{ and } p' \in \mathbb{B}^{\#}, \\ p & \text{otherwise,} \end{cases}$$

but any p such that $x = U(p')$ for all $p' \succeq p$ must have $|p| > n$. (It is easy to construct instances where the last inequality is equality.)

4.2.3 Dominance from simulation

Every simulation of one process on another establishes a dominance relation between the semimeasures generated by those processes. This is the key result for comparing the performance of different semimeasures, and for constructing universal semimeasures.

Theorem 4.8. For any pair of processes M and U we have

$$\mathrm{Err}[\mu_U(x)] \leq \mathrm{Err}[\mu_M(x)] + \mathrm{S_U(M)} \quad \text{for all } x \in X^*,$$

and

$$\mathrm{Err}[\mu_U(x{\downarrow})] \leq \mathrm{Err}[\mu_M(x{\downarrow})] + \mathrm{S_U(M{\downarrow})} \quad \text{for all } x \in X^*.$$

Proof. If $\mathrm{S_U(M)} = \infty$ then the theorem is trivial. Otherwise, let f^* be a weak simulation of M on U with constant $\mathrm{S_U(M)}$ guaranteed by Definition 4.5. We have

$$\begin{array}{rcl} \mu_M(x) & = & \lambda(M^{-1}(\Gamma_x)) \\ & \leq & 2^{\mathrm{S_U(M)}}\lambda(f(M^{-1}(\Gamma_x))) \\ & \leq & 2^{\mathrm{S_U(M)}}\lambda(U^{-1}(\Gamma_x)) \\ & = & 2^{\mathrm{S_U(M)}}\mu_U(x). \end{array}$$

The first and last steps are by definition. The second step holds by Lemmata A.1 and A.2 from the appendix, since f is monotone, injective, and satisfies $|f(p)| \leq |p| + \mathrm{S_U(M)}$ for all $p \in \mathbb{B}^\#$, and as $\lambda(A \cap \mathbb{B}^\infty) = \lambda A$ for any measurable set A. The third step holds because if $q \in f(M^{-1}(\Gamma_x))$ then $q = f(p)$ for some $p \in \mathbb{B}^\#$ with $x \preceq M(p)$. But by definition of weak simulation this means $x \preceq U(q)$, so $q \in U^{-1}(\Gamma_x)$.

The second half is similar. If $\mathrm{S_U(M{\downarrow})} = \infty$ then the theorem is trivial. Otherwise, let f^* be a strong simulation of M on U with constant $\mathrm{S_U(M{\downarrow})}$ guaranteed by Definition 4.5. We have

$$\begin{array}{rcl} \mu_M(x{\downarrow}) & = & \lambda(M^{-1}(\{x\})) \\ & \leq & 2^{\mathrm{S_U(M{\downarrow})}}\lambda(f(M^{-1}(\{x\}))) \\ & \leq & 2^{\mathrm{S_U(M{\downarrow})}}\lambda(U^{-1}(\{x\})) \\ & = & 2^{\mathrm{S_U(M{\downarrow})}}\mu_U(x{\downarrow}). \end{array}$$

The first and last steps are by definition. The second step holds by Lemmata A.2 and A.1. The third step holds because if $q \in f(M^{-1}(\{x\}))$ then $q = f(p)$ for some $p \in \mathbb{B}^\#$ with $x = M(p)$. But by definition of strong simulation this means $x = U(q)$, so $q \in U^{-1}(\{x\})$. $\square$

Dominance of arbitrary enumerable semimeasures

The above result extends simply to dominance of arbitrary enumerable semimeasures by Definition 4.6.

Corollary 4.9. Let U be a process, and ν be an enumerable semimeasure. The semimeasure μ_M weakly dominates ν with constant $\mathrm{S_M}(\nu)$:

$$\mathrm{Err}[\mu_U(x)] \leq \mathrm{Err}[\nu(x)] + \mathrm{S_U}(\nu) \quad \text{for all } x \in X^*,$$

and it strongly dominates ν with constant $\mathrm{S_M}(\nu\downarrow)$:

$$\mathrm{Err}[\mu_U(x\downarrow)] \leq \mathrm{Err}[\nu(x\downarrow)] + \mathrm{S_U}(\nu\downarrow) \quad \text{for all } x \in X^*.$$

Sequence complexity is an upper bound for error

Similarly, by Definition 4.7 we can extend to dominance of sequences. This shows the complexity of the true sequence is an upper bound for the semmimeasure's error. This allows us to view performance from the perspective of compression, which we do in Section 4.2.5.

Corollary 4.10. Let U be a process. We have

$$\mathrm{Err}[\mu_U(x)] \leq \mathrm{S_U(x)} \quad \text{for all } x \in X^*,$$

and

$$\mathrm{Err}[\mu_U(x\downarrow)] \leq \mathrm{S_U(x\downarrow)} \quad \text{for all } x \in X^*.$$

Proof. Apply Theorem 4.8 with $M = D_x$, the constant process defined by $D_x(p) = x$. Note that $\mu_{D_x}(x) = \mu_{D_x}(x\downarrow) = 1$ and so $\mathrm{Err}[\mu_{D_x}(x)] = \mathrm{Err}[\mu_{D_x}(x\downarrow)] = 0$. $\square$

4.2.4 Universal semimeasures from universal processes

A process U's semimeasure dominates the semimeasures of all processes it simulates, with constant at most the cost of simulation. A universal process, one which simulates all other processes, would thus dominate the semimeasures of all processes. Since every enumerable semimeasure is generated by some process, this semimeasure will dominate all enumerable semimeasures.

With this observation, we can finally construct a universal semimeasure.

Example 4.11. Take an enumeration M_i of processes (Theorem 3.25 constructs one). A prefix-free encoding $e\colon \mathbb{N} \to \mathbb{B}^*$ is one where given any p there is at most one i such that $e(i) \preceq p$. (Equivalently, $e(\mathbb{N})$ is a prefix-free set, and e is injective.) For example,

$$e_1(i) = 0^i 1$$

is such an encoding.

Define the process U by

$$U(p) = \begin{cases} M_i(q) & \text{if } p = e(i)q \text{ for some } i \in \mathbb{N} \text{ and } q \in \mathbb{B}^{\#}, \\ \epsilon & \text{otherwise.} \end{cases}$$

for all $p \in \mathbb{B}^{\#}$. The process U is well-defined because e is a prefix-free encoding.

The process U simulates all processes, for any process M equals the process M_i for some i, and

$$f(p) = e(i)p$$

is a simulation of M_i on U with constant $|e(i)|$. As a consequence

$$\mathrm{S_U(M) \leq S_U(M{\downarrow}) = S_U(M_i{\downarrow}) \leq |e(i)|.}$$

$\square$

This example generalises to Theorem 4.14 below.

Definition 4.12. A **weakly universal process** U is one which weakly simulates all processes. Similarly, a **strongly universal process** U strongly simulates all process. $\square$

Definition 4.13. A **weakly universal semimeasure** (or **universal semimeasure**) ξ is one which weakly dominates all enumerable semimeasures. Similarly, a **strongly universal semimeasure** ξ strongly dominates all enumerable semimeasures. $\square$

Theorem 4.14. If U is a weakly universal process, then μ_U is a weakly universal semimeasure. Similarly for strongly universal process and semimeasure.

Proof. Apply Theorem 4.8. □

Theorem 4.8 and Corollaries 4.9 and 4.10 can both be used to establish upper bounds[3] on a universal semimeasure's error. Theorem 4.8 says the universal semimeasure μ_U performs as well as any probabilistic computable model (see Section 3.4.6) but for a penalty equal to the complexity of the model with respect to U. Corollary 4.9 establishes that it performs as well as any enumerable semimeasure but for a penalty equal to the complexity of the semimeasure with respect to U. This shows to perform significantly better than a universal predictor, another predictor or probabilistic model must be significantly more complex.

We can establish an absolute bound on error too. By Corollary 4.10 above, the predictor μ_U's error is bounded above by the simulation complexity $\mathrm{S_U(x\downarrow)}$ of the true sequence $x \in X^{\#}$. The shorter the program $p \in \mathbb{B}^*$ for which $U(p') = x$ for all $p' \succeq p$, the lower $\mathrm{S_U(x\downarrow)}$ is. This means the more U can compress x the better μ_U performs.

The following, and final, section discusses why we might expect the sequences humans come across to be simple, relative to a sufficiently nice universal process U. Were this so, then μ_U would predict real-world sequences absolutely well.

4.2.5 Humanly accessible sequences may be simple

The previous section showed the more U can compress x, the better the universal semimeasure μ_U performs. This usage of "compression" is more general than that used in real-word compression problems. Decompression can use arbitrary amounts of space and time, whereas real-world algorithms cannot. In addition we need only supply the decompression algorithm: the predictor will, in effect, decompress all possible strings to form its predictions.

Suppose we want to compress a movie losslessly. An uncompressed movie is about 1 terabit (10^{12} bits) in size[4]. There are a number of standard compression methods.

1. The simplest method is not to compress it. To do this we prefix a short "echo" program, along with the size of the sequence, to the movie's bits. This results

[3]These bounds are exact i.e. no extra additive constants.

[4]$640 \times 480 \times 24 \times 24 \times 60 \times 100 \approx 10^{12}$bits.

in a slightly larger file (40bits plus the length of the echo program). We need to include the size of the sequence because our programs must know their own length. This is because we must have $U(p') = x$ for all $p' \succeq p$ where p is the program.

2. A general purpose compression algorithm such as zip may produce a shorter algorithm, although these methods tend to perform poorly on video data [Sal00]. Similar to before, we will have a program of the form dc where d is the decompression algorithm and c is the compressed data. All our compression methods are of this form.

3. A lossless video compression algorithm will perform better as the algorithms are specialised to video data. For example, in the related domain of audio encoding, lossless audio compression techniques can decrease filesize by over 50% [Lie04].

Given the unusual circumstances (i.e. unbounded space and time, the central importance of short programs) a number of more esoteric methods open up. These are necessarily speculative because these algorithms cannot be run in the real world, at least not in their naive form.

One such method is simulating the entire universe, then using the results of this simulation to generate a sequence. Simulating the universe requires the laws of physics and a lot of computational power, but we can use as much computing power as we need. If the laws of physics are compact this may require only a very short program (see [Teg96] for why the universe may be simple). This is in many ways analogous to computational theory proofs which run all possible programs in order to simplify the derivation (see e.g. [LV97] [Hut05]). That the universe as a whole may be simple, even though it contains complex parts, is analogous to the fact that the set of natural numbers is simple, even though particular numbers can be arbitrarily complex.

Simulating the universe not as strange as it may sound. The one thing we know about any sequence we encounter is that we encountered it within the universe[5]. This means it must appear somewhere in the simulation of the universe, near wherever humans appear in the simulation. The trick is finding short programs which can

[5]This kind of reasoning, called anthropic reasoning, is not as trivial as it may sound; see [Bos02] for details.

locate and extract these sequences. The length of such a program is difficult to estimate: we know little about the behaviour of extremely long running programs as we are not patient enough to wait for them to finish. In what follows we speculate on several possible methods.

1. One possible program could simulate the universe and extract the movie from someone's computer. To extract the movie we need to store the location of the computer (in time and space) and describe how the bits are stored inside the computer. It is difficult to estimate how much information this would require.
2. Instead of storing the location of the computer and instructions for how to extract the file, would could store a pattern to search for. The search process would comb the state of the universe looking for bitstreams matching the pattern, outputting the first that is found. For instance, if we try all relatively short ways of converting atoms into bitstreams, the first bitstream of length 10^{12} which has the same 10^9 first bits as the movie it finds may be the actual movie. Or, the first bitstream of length 10^{12} with the same (sufficiently long) hash code.
3. We could describe the movie as the nth movie filmed in human history. A similar method to the above could be used to decode this: a pattern searcher combing the universe first for human civilisation, then for its movies. Although storing n would require little space, the shortest pattern which detects human movies may be long. It is again difficult to estimate.

Future work

Theorem 3.29 shows semimeasures of processes are exactly enumerable semimeasures. Theorem 4.8 shows simulation between processes implies dominance between their semimeasures. Perhaps a converse could be proven, that whenever one enumerable semimeasure ν dominates another ρ there are processes U and M such that $\nu = \mu_U$, $\rho = \mu_M$, and U simulates M.

Extending the above, we show the semimeasure of universal processes are universal enumerable semimeasures. Is the converse true?

Section 4.2.2 introduced the simulation complexity $\mathrm{S_U(M)}$ of a process M relative to another U. It is easy to see

$$\mathrm{S_U(x) = S_U(D_x)}$$

is equal to the monotone complexity $Km_U(x)$ of x, where D_x is the constant process defined by $D_x(y) = x$. What does the strong simulation complexity $\mathrm{S_U(D_x{\downarrow})}$ correspond to? In general this complexity measure could do with future study.

Finally, the implications of replacing weak dominance with strong could do with investigation.

Conclusion

This thesis constructed, in detail, universal (enumerable) semimeasures. Universal semimeasures μ_U perform no worse than other other enumerable semimeasure ν but for a penalty of at most the complexity $\mathrm{S_U}(\nu)$ of ν relative to the underlying universal process U. Their error when predicting a sequence $x \in X^\#$ is at most the complexity $\mathrm{S_U(x{\downarrow})}$ of the sequence relative to U. For standard universal processes, e.g. those produced from common programming languages, the sequences we come across may be fairly simple making μ_U an absolutely powerful predictor.

Appendix A

Auxiliary lemmata

We prove two auxiliary lemmata for Theorem 4.8. Note that

$$\lambda(A) = \lambda(A \cap \mathbb{B}^\infty)$$

for any measurable set $A \subseteq \mathbb{B}^\#$. This is because $\lambda(\mathbb{B}^*) = 0$. For simplicity we will restrict our measure to subsets of $\mathbb{B}^\infty$, defining the cylinder set $\Gamma_l^0 = \{lp : p \in \mathbb{B}^\infty\}$.

Lemma A.1. If for some $x \in X^*$ and some process $M \colon \mathbb{B}^\# \to X^\#$, either $A = M^{-1}(\Gamma_x) \cap \mathbb{B}^\infty$ or $A = M^{-1}(\{x\}) \cap \mathbb{B}^\infty$, then

$$A = \bigcap_{i \in \mathbb{N}} \bigcup_{l \in L_i} \Gamma_l^0$$

for some sequence of sets $L_i \subseteq \mathbb{B}^*$ and where $\Gamma_l^0 = \{lp : p \in \mathbb{B}^\infty\}$.

Proof. If $x \preceq M(p)$ then by continuity of M there exists a finite prefix $p_f \prec p$ with $x \preceq M(p_f)$. By monotonicity of M we have $\Gamma_{p_f} \subseteq M^{-1}(\Gamma_x)$. Thus,

$$M^{-1}(\Gamma_x) = \bigcup_{p_f \in \mathbb{B}^*} \Gamma_{p_f}[x \preceq M(p_f)]$$

and so

$$M^{-1}(\Gamma_x) \cap \mathbb{B}^\infty = \bigcup_{p_f \in \mathbb{B}^*} \Gamma_{p_f}^0[x \preceq M(p_f)]$$

since $\Gamma_{p_f}^0 = \Gamma_{p_f} \cap \mathbb{B}^\infty$.

Note that

$$\mathbb{B}^\infty \setminus \Gamma_l^0 = \bigcup_{\substack{p \neq l \\ |p|=|l|}} \Gamma_p^0$$

for any $l \in \mathbb{B}^*$. Thus

$$\begin{aligned}
M^{-1}(\{x\}) \cap \mathbb{B}^\infty &= M^{-1}(\Gamma_x \setminus \cup_{c\in X}\Gamma_{xc}) \cap \mathbb{B}^\infty \\
&= \big(M^{-1}(\Gamma_x)) \cap \mathbb{B}^\infty\big) \setminus \big(M^{-1}(\cup_{c\in X}\Gamma_{xc}) \cap \mathbb{B}^\infty\big) \\
&= \left(\bigcup_{l\in L} \Gamma_l^0\right) \setminus \bigcup_{l\in L'} \Gamma_l^0 \\
&= \left(\bigcup_{l\in L} \Gamma_l^0\right) \cap \bigcap_{l\in L'} (\mathbb{B}^\infty \setminus \Gamma_l^0) \\
&= \left(\bigcup_{l\in L} \Gamma_l^0\right) \cap \bigcap_{l\in L'} \left(\bigcup_{\substack{p \neq l \\ |p|=|l|}} \Gamma_p^0\right)
\end{aligned}$$

for some sets $L, L' \subseteq \mathbb{B}^*$. □

Lemma A.2. Let $f\colon \mathbb{B}^{\#} \to \mathbb{B}^{\#}$ be a monotone injective function satisfying

$$|f(p)| \leq |p| + k \quad \text{for all } p \in \mathbb{B}^{\#}$$

for some $k \in \mathbb{N}$. If

$$A = \bigcap_{i\in\mathbb{N}} \bigcup_{l \in L_i} \Gamma^0_l$$

for $L_i \subseteq X^*$ and where $\Gamma^0_l = \{lp : p \in \mathbb{B}^\infty\}$, then

$$\lambda(A) \leq 2^k \lambda(f(A)).$$

Proof. First, we have

$$\begin{aligned}
\lambda(\Gamma^0_l) &\overset{(1)}{=} \lim_{j\to\infty} \sum_{p_j \in \mathbb{B}^j} \lambda(\Gamma^0_{lp_j}) \\
&\overset{(2)}{\leq} 2^k \lim_{j\to\infty} \sum_{p_j \in \mathbb{B}^j} \lambda(\Gamma^0_{f(lp_j)}) \\
&\overset{(3)}{=} 2^k \lambda\left(\bigcap_j \bigcup_{p_j \in \mathbb{B}^j} \Gamma^0_{f(lp_j)} \right) \\
&\overset{(4)}{=} 2^k \lambda(f(\Gamma^0_l))
\end{aligned}$$

for any $l \in \mathbb{B}^*$. The following justifies each step.

1. We have
$$\Gamma^0_l = \bigcap_i \Gamma^0_l = \bigcap_i \bigcup_{p_i \in \mathbb{B}^i} \Gamma^0_{lp_i}$$
since if $x \in \mathbb{B}^\infty$ begins with l then for any i it begins with lp_i for some $p_i \in \mathbb{B}^i$. As $\Gamma^0_l \subseteq \Gamma^0_l$ we can exchange intersection with limit. As $\Gamma^0_{lp_i}$ for $\mathbb{B}^i$ are pairwise disjoint for fixed i, we can exchange disjoint union with summation.

2. This holds because $|f(p)| - k \leq |p|$, and as $\lambda(\Gamma^0_p) = 2^{-|p|}$.

3. We have,

$$\lim_{j\to\infty} \sum_{p_j \in \mathbb{B}^j} \lambda(\Gamma^0_{f(lp_j)}) = \lim_{j\to\infty} \lambda\left(\bigcup_{p_j \in \mathbb{B}^j} \Gamma^0_{f(lp_j)} \right) = \lambda\left(\bigcap_j \bigcup_{p_j \in \mathbb{B}^j} \Gamma^0_{f(lp_j)} \right)$$

as f is injective, so $\Gamma^0_{f(lp_j)}$ are pairwise disjoint for fixed j, and as f is monotone, so $\bigcup_{p_{j+1}\in\mathbb{B}^{j+1}}\Gamma^0_{f(lp_{j+1})} \subseteq \bigcup_{p_j\in\mathbb{B}^j}\Gamma^0_{f(lp_j)}$ for all j.

4. Observe that
$$f(\Gamma^0_l) = \bigcap_j \bigcup_{p_j\in\mathbb{B}^j} \Gamma^0_{f(lp_j)}.$$
To see this, suppose $x \in \mathbb{B}^\infty$ is in the left hand set. Then $x = f(lp)$ for some $p \in \mathbb{B}^\infty$. For every j there exists $p_j \in \mathbb{B}^j$ such that $p_j \prec p$, and as f is monotone $f(lp_j) \preceq f(lp) = x$. So x is in the right hand set.

   Suppose instead that $x \in \mathbb{B}^\infty$ is in the right hand set. Then for every j there exists $p_j \in \mathbb{B}^j$ such that $f(lp_j) \preceq x$. Since the prefixes of x form a linear order, and as f is monotone and injective, we have $lp_j \preceq lp_{j+1}$ and $f(lp_j) \preceq f(lp_{j+1})$ for all j. So $\lim_{j\to\infty} lp_j$ is well-defined and equal to lp for some $p \in \mathbb{B}^\infty$, and $\lim_{j\to\infty} f(lp_j)$ is well-defined, with $\lim_{j\to\infty} f(lp_j) \preceq x$.

   We have $\lim_{j\to\infty} f(lp_j) = x$, since as f is injective the limit is an infinite sequence. But then as $\lim_{j\to\infty} f(lp_j) \preceq f(lp)$ we have $x = f(lp)$. So x is in the right hand set.

Next, observe that

$$\begin{aligned} A &= \bigcap_i \bigcup_{l\in L_i} \Gamma^0_l \\ &= \bigcap_i \left(\bigcap_{j\le i} \bigcup_{l\in L_j} \Gamma^0_l \right) \\ &= \bigcap_i \bigcup_{l\in B_i} \Gamma^0_l \end{aligned}$$

for some $B_i \subseteq \mathbb{B}^*$ since sets of the form $\cup_{l\in L_i}\Gamma^0_l$ are closed under finite intersection (see the proof of Lemma 2.7). We replace L_i with B_i to ensure that

$$\bigcup_{l\in B_{i+1}} \Gamma^0_l \subseteq \bigcup_{l\in B_i} \Gamma^0_l$$

holds for all i (i.e. that $\cup_{l\in B_i}\Gamma^0_l$ is decreasing in i), to allows us to swap intersections with limits below. We can assume B_i are prefix-free since

$$\bigcup_{l\in B_i} \Gamma^0_l = \bigcup_{l\in \min B_i} \Gamma^0_l$$

and $\min B_i$ is prefix-free.

Finally, we have

$$\begin{aligned}
\lambda(A) &\overset{(1)}{=} \lambda\left(\bigcap_{i\in\mathbb{N}}\bigcup_{l\in B_i}\Gamma_l^0\right)\\
&\overset{(2)}{=} \lim_{i\to\infty}\sum_{l\in B_i}\lambda(\Gamma_l^0)\\
&\overset{(3)}{\le} 2^k\lim_{i\to\infty}\sum_{l\in B_i}\lambda(f(\Gamma_l^0))\\
&\overset{(4)}{=} 2^k\lambda\left(\bigcap_{l}\bigcup_{l\in B_i}f(\Gamma_l^0)\right)\\
&\overset{(5)}{=} 2^k\lambda(f(A)).
\end{aligned}$$

The following justifies each step.

1. By the Lemma's hypothesis, and by definition of B_i.

2. Since $\bigcup_{l\in B_{i+1}}\Gamma_l^0 \subseteq \bigcup_{l\in B_i}\Gamma_l^0$ by definition of B_i, we can exchange the intersection for a limit. As B_i is prefix-free, we can exchange the disjoint union with summation.

3. Because $\lambda(\Gamma_l^0) \le 2^k\lambda(f(\Gamma_l^0))$.

4. Observe that

$$\lim_{i\to\infty}\sum_{l\in B_i}\lambda(f(\Gamma_l^0)) = \lim_{i\to\infty}\lambda\left(\bigcup_{l\in B_i}f(\Gamma_l^0)\right) = \lambda\left(\bigcap_{l}\bigcup_{l\in B_i}f(\Gamma_l^0)\right).$$

   First step is because $f(\Gamma_l^0)$ are pairwise disjoint for $l\in B_i$ and fixed i, since B_i is prefix-free and f is injective. Second step is because

$$\bigcup_{l\in B_i}f(\Gamma_l^0) = f\left(\bigcup_{l\in B_i}\Gamma_l^0\right)$$

   is decreasing in i as $\bigcup_{l\in B_i}\Gamma_l^0$ is.

5. We have

$$\bigcap_i \bigcup_{l\in B_i} f(\Gamma_l^0) = \bigcap_i f\left(\bigcup_{l\in B_i} \Gamma_l^0\right) = f\left(\bigcap_i \bigcup_{l\in B_i} \Gamma_l^0\right) = f(A).$$

First step holds for any function f, but the second holds because f is injective.

□

Bibliography

[Ash72] Robert B. Ash. *Real analysis and probability.* Academic Press, New York, 1972. Probability and Mathematical Statistics, No. 11.

[Ber79] Jose-M. Bernardo. Expected information as expected utility. *Ann. Statist.*, 7(3):686–690, 1979.

[Ber93] James O. Berger. *Statistical decision theory and Bayesian analysis.* Springer Series in Statistics. Springer-Verlag, New York, 1993. Corrected reprint of the second (1985) edition.

[Bos02] Nick Bostrom. *Anthropic Bias: Observation Selection Effects in Science and Philosophy (Studies in Philosophy).* Routledge, 2002.

[Cal02] Cristian S. Calude. *Information and randomness.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, second edition, 2002. An algorithmic perspective, With forewords by Gregory J. Chaitin and Arto Salomaa.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley Series in Telecommunications. John Wiley & Sons Inc., New York, 1991. A Wiley-Interscience Publication.

[Dud89] Richard M. Dudley. *Real analysis and probability.* The Wadsworth & Brooks/Cole Mathematics Series. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA, 1989.

[Fre90] Edward Fredkin. Digital mechanics: an informational process based on reversible universal cellular automata. *Phys. D*, 45(1-3):254–270, 1990. Cellular automata: theory and experiment (Los Alamos, NM, 1989).

[Gan80] Robin Gandy. Church's thesis and principles for mechanisms. In *The Kleene Symposium (Proc. Sympos., Univ. Wisconsin, Madison, Wis., 1978)*, volume 101 of *Stud. Logic Foundations Math.*, pages 123–148. North-Holland, Amsterdam, 1980.

[GR04] Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. Technical Report 463, Department of Statistics, University of Washington, 2004.

[Gur05] Yuri Gurevich. Interactive algorithms 2005. In *Mathematical foundations of computer science 2005*, volume 3618 of *Lecture Notes in Comput. Sci.*, pages 26–38. Springer, Berlin, 2005.

[HH99] John Hamal Hubbard and Barbara Burke Hubbard. *Vector calculus, linear algebra, and differential forms.* Prentice Hall Inc., Upper Saddle River, NJ, 1999. A unified approach.

[Hut05] Marcus Hutter. *Universal artificial intelligence.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2005. Sequential decisions based on algorithmic probability.

[Jay68] Edwin T. Jaynes. Prior probabilities. *IEEE Trans. on Systems Science and Cybernetics*, SSC-4(5):227, 1968.

[Jay03] E. T. Jaynes. *Probability theory.* Cambridge University Press, Cambridge, 2003. The logic of science, Edited and with a foreword by G. Larry Bretthorst.

[Knu92] Donald E. Knuth. Two notes on notation. *Amer. Math. Monthly*, 99(5):403–422, 1992.

[KST82] Daniel Kahneman, Paul Slovic, and Amos Tversky. *Judgment under Uncertainty: Heuristics and Biases.* Cambridge University Press, 1982.

[Leg97] Shane Legg. Solomonoff induction. Technical Report 30, Centre for Discrete Mathematics and Theoretical Computer Science. University of Auckland, 1997.

[Lie04] Tilman Liebchen. An introduction to MPEG-4 audio lossless coding. In *IEEE ICASSP 2004, Montreal, May 2004*, 2004.

[LV97] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications.* Graduate Texts in Computer Science. Springer-Verlag, New York, second edition, 1997.

[Odi89] Piergiorgio Odifreddi. *Classical recursion theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, 1989. The theory of functions and sets of natural numbers, With a foreword by G. E. Sacks.

[Pol04] Jan Poland. A coding theorem for enumerable output machines. *Inform. Process. Lett.*, 91(4):157–161, 2004.

[Sal00] D Salomon. *Data compression: the complete reference.* New York: Springer, 2nd edition, 2000.

[Sch02] Jürgen Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *Internat. J. Found. Comput. Sci.*, 13(4):587–612, 2002.

[Sol64] R. J. Solomonoff. A formal theory of inductive inference. parts I and II. *Information and Control*, 7:1–22 and 224–254, 1964.

[Sol78] R. J. Solomonoff. Complexity-based induction systems: comparisons and convergence theorems. *IEEE Trans. Inform. Theory*, 24(4):422–432, 1978.

[Teg96] Max Tegmark. Does the universe in fact contain almost no information? *Found. Phys. Lett.*, 9(1):25–41, 1996.

[ZL70] A. K Zvonkin and L. A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6):83–124, 1970.